Sistemas de Construcción de Mapas en PDDL para la Planificación Automática.

PROYECTO FIN DE GRADO

Departamento de Inteligencia Artificial

Félix Caro Herranz

Autor: Félix Caro Herranz.
Tutor: Moisés Martínez Muñoz.

Título: Sistemas de Construcción de mapas en PDDL para la Planificación Automática.

Autor: Félix Caro Herranz.

Director: Moisés Martínez Muñoz.

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Resumen

Uno de los principales objetivos de la robótica, es diseñar sistemas en los cuales diferentes robots con distintas características y funcionalidades, colaboren entre sí para resolver problemas más complejos. En ese proyecto, se presenta el diseño e implementación de un sistema de control que es capaz de generar mapas de entornos en formato XML, que puedan ser utilizados por ejemplo, por un sistema de planificación automática "planificador".

El objetivo principal, es que estos mapas puedan ser utilizados por otros robots que no tengan la capacidad o los dispositivos necesarios para su generación. Para la realización de este proyecto, se ha utilizado el framework ROS (Robot Operation System). Mediante el cual, se ha desarrollado un sistema de control para el robot P3DX, que utiliza una cámara Microsoft Kinect como láser para la recogida de información del entorno.

La recogida de datos se hará en entornos reales, como puede ser las salas o pasillos de la universidad.

Palabras clave: ROS, mapa, Microsoft Kinect, Waypoints.

Abstract

One of the robotics main goals is to design systems in which different robots with different characteristics and functionality collaborate to solve complex problems. This work presents the design and implementation of a control system which is able to generate environment maps in XML format. These maps could be used, for example, by automatic planning systems.

The main aim is to achieve that others robots without the ability of generating maps, could use this maps. In this work it has been used the ROS framework (Robot Operation System). With this framework it has developed a control system for the P3DX robot, which uses a Kinect as the laser to the environment information collection.

The data collection will be done in real environments, like University rooms and corridors.

Keywords: ROS, map, Microsoft Kinect, waypoints, PDDL, P3DX

Índice general

Índice de Ilustraciones

Índice de Tablas

Índice de Fórmulas

Capítulo 1. Introducción

En los últimos años, la robótica inteligente ha sufrido muchos avances que unido a la aparición de un gran número de robots, han generado un ambiente en el que es más sencillo construir sistemas colaborativos que permiten la realización de tareas más complejas.

Para la colaboración entre ellos, se necesita que la información que comparten, tenga una misma estructura que les permita entenderla y de la cual, puedan extraer la información necesaria para realizar sus tareas. Esta información deberá contener datos que sirva para planificar tareas y lograr sus objetivos.

El objetivo de crear sistemas colaborativos, es la de completar tareas que sistemas por separado no serían capaces de realizar. En este proyecto lo que se pretende, es crear mapas de entornos que sean utilizados por diferentes robots que no son capaces de generarlos por sí solos.

La información que recogen estos mapas son las zonas libres de obstáculos en las que puede llevar a cabo otro robot sus tareas. Estos mapas recogerán también la información necesaria para que los robots puedan desplazarse por ellos.

Estos mapas permitirán generar planes de desplazamiento, para que los robots puedan alcanzar las posiciones en las que se desea que trabajen.

En los siguientes puntos de este documento, se realizará un estudio del desarrollo de un sistema capaz de generar mapas para sistemas robóticos.

1.1 Definición del Problema a Tratar

Actualmente, existe un amplio número de robots que han sido diseñados con una serie de características que no les permiten realizar de forma correcta muchas de las posibles tareas que pueden llevar acabo de forma individual. Esto implica, que si se construyesen sistemas en los cuales diferentes robots fueran capaces de ofrecerse ayuda entre sí para resolver estos problemas, sería muy útil, ya que disminuiría el coste de los robots, debido a que se reduciría su complejidad y el coste computacional que conlleva realizarlo de forma individual. De forma más precisa, el problema a resolver podría describirse de la siguiente manera:

- Dado un conjunto de robots, con diferentes características, tomar un robot que tenga la capacidad de generar mapas y hacer que este genere los mapas para los demás.

Esto implica que al menos uno de los robots, debe ser capaz de navegar por el entorno y generar un mapa que pueda ser utilizado por el resto de robots para la realización de sus tareas.

En este proyecto, se pretende construir un sistema de control que permita a un robot P3DX construir mapas usando una Kinect, los cuales, puedan ser utilizados posteriormente por otros robots, por ejemplo para planificar. A continuación, se presentan de forma más precisa los diferentes problemas a resolver durante el desarrollo de este proyecto.

- Entornos Dinámicos**:** Comúnmente los entornos en los cuales trabajará el robot, serán dinámicos. Estos son entornos en los que existen elementos que se muevan libremente por él, como por ejemplo personas u otros robots. Esto aumenta la complejidad del proceso de generación de mapas, ya que dificulta la toma de referencias a la hora de recrear el entorno.

- Localización: El robot debe ser capaz de auto localizarse en el entorno, por lo cual, deberá basarse en las medidas de las distancias a los objetos que recogen sus sensores, así como tener constancia del recorrido que lleva realizado por el entorno.

- Navegación: El robot debe ser capaz de esquivar los obstáculos que se encuentra en el camino mientras recrea el entorno. Esto dependerá en gran medida del error que comentan los sensores que utiliza el robot para obtener información del entorno. Estos errores pueden ser de varios tipos:

    - Ruido sensorial: Este tipo de errores son debidos a la inferencia entre sensores de distintos dispositivos o debido a las características físicas de los elementos de un entorno. Este ruido puede reducir de forma drástica la utilidad de la información recogida por los sensores.

    - Aliasing sensorial: Este problema surge cuando el robot se cree que esta en un emplazamiento ya explorado y en realidad es otro sitio desconocido. Esto es debido a que los dos lugares comparte las mismas características captadas por los sensores.

    - Error de odometría: Este tipo de errores suelen aparecer cuando los distintos elementos utilizados para la recogida de información del entorno, no han sido calibrados correctamente, generando por ejemplo, falsos positivos.

Como se puede observar, el problema al que se quiere hacer frente, necesita del estudio de disciplinas como la electrónica, la física y las matemáticas. Esto hace que se requiera un estudio de las diferentes áreas, para lograr que el proyecto cumpla sus objetivos y tenga éxito.

1.2 Objetivos

A continuación, se presentan de forma detallada los objetivos que conforman el alcance de este proyecto.

El objetivo principal del programa que se quiere desarrollar, es el que dado un entorno desconocido, el robot P3DX sea capaz de determinar de manera automática que elementos o regiones de un entorno pueden ser navegables por el robot, y en base a estos crear una serie de marcas o puntos (Waypoints[1]) reconocibles por el robot. Una vez que se ha determinado que zonas son navegables, el robot deberá ser capaz de alcanzar uno de los waypoints del mapa.

Para lograr este objetivo se necesitará alcanzar las siguientes fases:

- Estudio y análisis de ROS: Se deberá realizar un estudio y análisis de framawork ROS para su compresión. Esto permitirá que el desarrollo del sistema sea más fácil y sencillo. Además, proporcionar distintas funcionalidades para la colaboración de sistemas.

- Diseño e implementación de un sistema de generación de mapas: Se deberá realizar un sistema capaz de recrear entornos, para ello se deberá cumplir con los siguientes objetivos.

    - Exploración del entorno: El robot P3DX debe navegar de forma autónoma por un entorno sin depender de las órdenes dadas por el ser humano. A pesar de esta restricción, podrá tener una navegación controlada (por el teclado del ordenador) de manera opcional. Cuando el robot navegue por el entorno, deberá ser capaz de alcanzar todas las zonas de éste. Se necesita que realice esta funcionalidad para poder construir el mapa del entorno completo.

    - Creación del mapa: El sistema deberá de crear un mapa en dos dimensiones en el que se muestren las zonas libres de obstáculos.

    - División en waypoints: Para que la información del entorno se útil a otros sistemas, se deberá crear una referencia topográfica en las zonas libres del entorno que contenga la información necesaria para un planificador automático.

    - Navegación autónoma en el entorno: El robot debe de ser capaz de navegar por el entorno basándose en el mapa creado y en los waypoints establecidos.

- Experimentación y evaluación del sistema desarrollado: El sistema deberá ser probado en diferentes escenarios para poder evaluar su comportamiento.

El servicio principal de la aplicación que se quiere construir, será el que se dedique al análisis de la imagen para conseguir un mapa de relevancia que indique sobre la imagen analizada, cuales son las zonas libres de obstáculos y asignarlas un Waypoint.

1.3 Motivaciones

En la actualidad, los sistemas robóticos se están alejando del desarrollo de tareas sistemáticas. Estos sistemas realizan tareas que antes ejecutaban los humanos de manera eficiente e incluso

[1] Waypoints. Conjunto de marcas en un entorno que sirven para fijar rutas o posicionamientos de un individuo.

mejor en algunos casos. Por eso, se está produciendo un fuerte crecimiento en desarrollo de estas tecnologías, haciendo que se oferten muchos puestos de trabajo.

La *Ilustración 1.3-1* muestra el crecimiento en los últimos años del parque robótico en España, según la Asociación de Robótica y Automatización de sistemas de Producción (1).

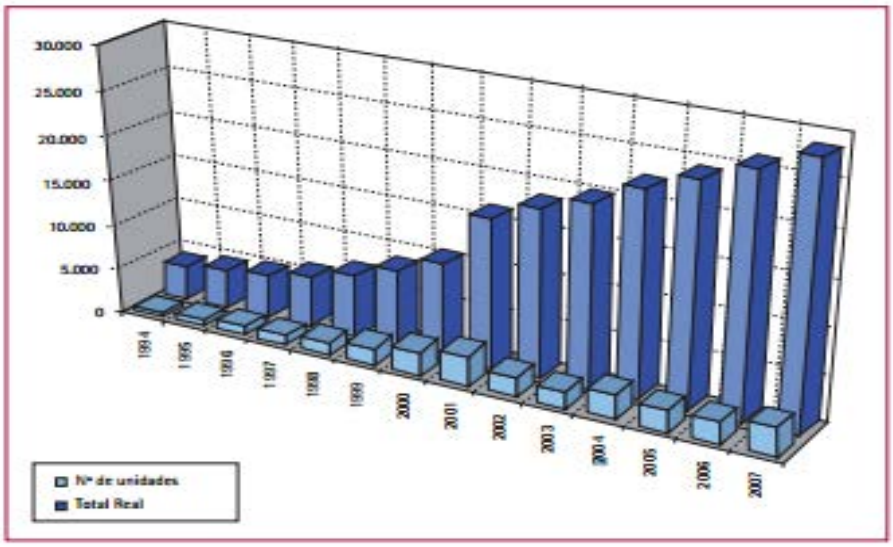

Ilustración 1.3-1. Crecimiento del parque robótico en España

Una de las motivaciones que me ha llevado a realizar este proyecto, ha sido la formación en diseño de sistemas robóticos, ya que es una industria que está creciendo mucho y pude ayudarme a la hora de incorpórame al mercado laboral.

Para la creación de sistemas robóticos, es necesario el conocimiento de varias áreas, como puede ser la física, la electrónica o la informática. Por eso elegí este proyecto, ya que quiero obtener más conocimientos sobre estas áreas y completar más mí formación.

Los sistemas robóticos son un tipo de sistemas que se enfrenta a muchas situaciones difíciles con problemas inesperados, por ello, otra motivación que me llevo a elegir este proyecto, fue mi afán a la hora de adquirir destreza en la solución de problemas de ingeniería.

La elaboración de este proyecto, me permitirá enfrentarme a diferentes situaciones que me pueden surgir a lo largo de mi vida profesional para las cuales podré afrontarlas con un mayor grado de conocimiento y destreza.

1.4 Estructura del Proyecto

Este documento se divide en siete capítulos, de los cuales el último alberga los distintos anexos del proyecto. A continuación, se realiza una descripción del contenido de cada uno de ellos:

- **Introducción**: En este capítulo, se hace una breve descripción del problema a tratar en este proyecto y de los objetivos que se quieren conseguir.

- **Estado de la Cuestión**: En este apartado del documento, se hace una introducción al estado en el que se encuentra la robótica móvil hoy en día y al problema de planificación, modelado y localización de robots. Se muestra las principales estrategias para el modelado, exploración y planificación de entornos con robots móviles. Se pretende dar cabida a posibles errores y situaciones que se puedan dar en el desarrollo del proyecto.

- **Análisis del Sistema:** Se realiza la exposición de una especificación detallada del sistema, que satisface los objetivos y una descripción que sirva de base para el posterior diseño de éste.

- **Diseño de la Solución Técnica:** En este apartado del documento, se explicará las pautas que se han seguido para el desarrollo del sistema

- **Pruebas realizadas al Sistema**: Este capítulo define el conjunto de pruebas a realizar sobre este sistema para evaluar el cumplimiento de los requisitos definidos en el apartado de Análisis de Sistema.

- **Planificación del Trabajo y Entorno Socio-Económico**: Pretende realizar un estudio de la organización para poder efectuar un desarrollo del proyecto tanto eficaz como eficiente.

- **Conclusiones y trabajos futuros**: En este capítulo, se describen las conclusiones generales obtenidas, así como el resultado de los experimentos realizados en capítulo anterior. También, se realiza una descripción de los trabajos que se podrían realizar basados en este proyecto. Por último, se hace una descripción de los problemas que han surgido durante la vida del proyecto.

- **Anexo I. Manual de Instalación**: En este anexo, se describen las pautas a seguir para poder llevar a cabo la instalación del proyecto.

- **Anexo II Manual de Usuario**: Este anexo contiene la descripción de las acciones que debe de realizar un usuario para la ejecución de todas las funcionalidades del sistema implementadas en este proyecto.

- **Anexo III ROS**: En este anexo del documento, se realiza una descripción del sistema operativo de robots utilizado en este proyecto. También, se describe las funcionalidades y herramientas que proporciona ROS.

- **Anexo IV. Hardware del Sistema**: En este anexo, se describen las características físicas del Robot P3DX y de la cámara Microsoft Kinect.

- **Anexo V. Diseño Detallado de la Aplicación**: En este anexo, se hace una descripción detallada de las partes del software que comprende el sistema implementado.

- **Anexo VI. Instalación de la Cámara Microsoft Kinect**: En este anexo del documento, se realiza una descripción de como instalar la cámara Microsoft Kinect a la corriente eléctrica del robot PDX3.

Capítulo 2. Estado de la Cuestión

Este capítulo con tiene información de los diferentes temas que van a ser tratados en el desarrollo de este proyecto. Esto permitirá realizar un estudio y ver las diferentes cuestiones que pueden surgir.

En esta parte, se verá una descripción de los problemas de localización, planificación automática, modelado de entornos, construcción de mapas y modelado del conocimiento con PDDL y una descripción de lo que es un marco de referencia.

Todo ello servirá de ayuda para sentar las bases del desarrollo del proyecto y ayudar a la compresión de ciertos temas de Planificación, Localización y Modelado.

Un robot móvil necesita navegar de forma robusta, por eso debe saber dónde se encuentra dentro de un entorno, así surge el problema de la localización. Para abordar este problema, se realiza la siguiente formulación:

- **Localización**: Dado un entorno descrito en forma de mapa, rejilla de ocupación o descripción geométrica, averiguar la posición en la que se encuentra el robot en el entorno a través de la información que captan sus sensores. Esta información suele ser la distancia con la que se encuentra de los diferentes obstáculos y la adquirida por sus desplazamientos dentro del mapa (2).

El problema de la localización está ligado a los problemas de planificación y mapeado de un entorno. Al igual que el problema de la localización, se hará una formulación para el problema del mapeado y planificación respectivamente. La formulación del problema del mapeado es la siguiente:

- **Modelado**: Dado una serie de observaciones realizadas por un robot, construir un modelo del entorno que pueda ser utilizado por algoritmos de localización y planificación (3).

La formulación del problema de planificación es la siguiente:

- **Planificación**: Dado un mapa de un entorno y la posición del robot en ese entorno, alcanzar otra posición distinta a la inicial de la manera más eficiente posible (4).

Los modelos propuestos para la resolución de los problemas de localización, mapeado y planificación deben de tener en cuenta las limitaciones que tiene un robot móvil que son las siguientes:

- **Actuación en Tiempo Real**: El robot debe actuar en tiempo real cuando recibe un estímulo, el problema que surge es que el robot necesita de un cierto rango de tiempo para tomar una decisión frente a ese estimulo.

- **Entornos dinámicos**: Cuando se pone a navegar a un robot en un entorno, éste toma referencias de los objetos del entorno, estos objetos son cambiantes por lo cual, el robot puede realizar cálculos erróneos cuando hay modificación en los objetos.

- **Ruido en la posición**: Los movimientos del robot no suelen ser exactos produciéndose errores en la odometría, estos errores son de tipo acumulativo y tiene como consecuencia errores graves en el cálculo posicionamiento del robot y la estimación de los movimientos.

- **Ruido en la información de los sensores**: La información captada por los sensores no es totalmente exacta habiendo pequeñas variaciones, por lo cual esto causa los mismos efectos que el ruido en el posicionamiento del robot.

2.1 Modelado de un Entorno

En este apartado se realizará un estudio de los tipos de mapas que utiliza un robot móvil para modelar el entorno. Hay diferentes tipos de entorno y se agrupan según el área de trabajo y según los objetos presentes en el entorno (5).

- **Según el área de trabajo**: Se distinguen dos tipos de entornos, el interior y el exterior. El interior está claramente delimitados por paredes y cielorrasos, además suelen tener luz artificial. Por el contrario los exteriores no están delimitados por ninguna estructura.

- **Según los objetos del entorno**: Pueden ser dinámicos o estáticos. Los entornos estáticos son aquellos que los objetos no cambia de posición ni de estructura, suelen tener siempre las mismas características físicas y se les atribuye una forma geométrica. En cambio los entornos dinámicos son aquellos en los que los objetos son cambiantes tanto de posición como de forma.

El objetivo de realizar un modelado del entrono no es otro que el de proporcionar elementos comunes a las tareas de localización y planificación. Las tareas de localización y planificación comparan la información obtenida por los sensores del robot y la información recogida en el mapa, la actualización de la posición del robot dependerá de esta comparación (6).

Los criterios utilizados para la comparación de los diferentes tipos de mapas serán los siguientes:

- **Asociación de datos:** La información que recoge el mapa debe de ser los más real posible, por eso se hace una comparación de los datos que recoge el robot y la información almacenada en el mapa. Para que este proceso sea lo más eficiente, se debe asegura de que la información sea robusta para poder obtener un buen funcionamiento en caso de exploración de grades entornos.

- **Convergencia monótona:** Se dice que hay convergencia en la creación de un mapa, cuando la geometría de los objetos que describe el mapa y la geometría de los objetos reales, se aproxima cada vez más con cada nueva exploración.

- **Detención de un ciclo:** Esto es la capacidad que tiene un robot cunado está explorando para detectar un lugar ya explorado. La robustez en la decisión de si una asociaciones correcta o no es muy importante en la creación de un mapa, por eso el error acumulado en la creación de un mapa debe ser compensado en cada exploración.

- **Representación de la incertidumbre:** La localización deriva del mapa, esto conlleva cierto de grado de incertidumbre debido al instrumental utilizado para la creación del mapa. Por eso, un modelo de incertidumbre deberá mostrar con fidelidad la incertidumbre que existe con el entorno real y el mapeado.

En los siguientes apartados se verán lo diferentes métodos que hay para el modelado de un entorno.

2.1.2 Mapas Topológicos

En robótica se consideran diferentes métodos topológicos para representar el entorno. Estos métodos se pueden clasificar desde el punto de vista del tipo de información que contienen y las relaciones que se derivan.
Existen modelos que carecen de información geométrica, donde los elementos son representados como (marca a la derecha de la puerta, puerta izquierda, etc.), y requieren de sistemas de percepción e identificación sofisticados y fiables para la navegación.

Los mapas topológicos realizan una descripción del entorno basándose en grafos[2]. Los nodos representan intersecciones o entradas, marcas, objetivos, etc. Los arcos representan
Un camino navegable entre dos nodos, estableciendo una relación espacial.
Los arcos pueden contener información adicional como: dirección (Norte, Sur, Este, Oeste), distancias, tipo de terreno, comportamiento requerido para navegar, etc.

Para la creación de estos mapas, se utiliza el algoritmo SLAM. Este algoritmo inserta un nodo en el grafo cuando encuentra una zona nueva del entorno que no ha sido explorada, este nodo representa un sito del mapa (6). El algoritmo sigue insertando nodos a medida que el robot se desplaza por el entorno. Cuando un robot reconoce un sito que ya ha sido guardado y tiene un nodo en el grafo, termina o hace que el robot modifique su trayectoria para la búsqueda de nuevos espacios. La *Ilustración 2.1-1* muestra un mapa topológico.

[2] Grafo. Conjunto de objetos (nodos y arcos) que permiten representar relaciones binarias entre elementos de un conjunto.

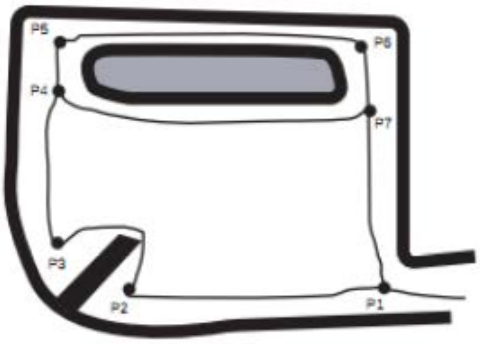

Ilustración 2.1-1. Mapa Topológico

La navegación topológica depende de la presencia y reconocimiento de las marcas, una marca, un espacio marcado del entorno modelado. Las marcas pueden ser un objeto auto-contenido como una baliza, o un grupo de objetos con algún significado, en ambos casos, el sensor en cargado de la percepción debe ser capaz de reconocerla. Si un robot móvil es capaz de localizar una marca en el entorno y esa marca se encuentra dentro del mapa utilizado por el robot, entonces el robot está localizado con respecto a ese mapa. Un caso frecuente de estudio es colocar una serie de marcas reconocibles por el sistema de percepción en el espacio de trabajo.

La ventaja que tiene este tipo de mapas es que proporción una buena información para realizar la interfaz con procesos de planificación.

El problema que surge con este tipo de mapas es de Asociación de Datos, ya que es difícil determinar por los mecanismos del robot si se ha encontrado un espacio nuevo o un espacio que ya ha sido marcado. Este tipo de errores puede hacer que el robot pierda su localización.

2.1.3 Mapas de Ocupación de Celdilla

Este mapa consiste en una división de celdas del espacio explorado. Una rejilla de ocupación es una discretización del entorno a modelar en celdas. Dicha discretización vendrá dada por el tamaño del mundo en el que se encuentra el robot. Para la representación del problema se utilizará una matriz cuyas celdas contendrán un valor de probabilidad, el cual indica la certidumbre de que en esa posición, haya un obstáculo (1.0) o exista espacio libre (0.0). Al comienzo de la creación del mapa, a las celdillas se les asignará un valor de (0.5). Esta probabilidad está condicionada por la lectura de los sensores de medición que utiliza el robot para medirse la distancia con los respectivos objetos, a mayor distancia, mayor probabilidad de dar un caso erróneo. También hay que tener en cuenta el posible error que pueda tener el sensor de medición, a este error se le dará un valor fijo a priori (7). La *Ilustración 2.1-2* muestra un mapa de ocupación de celdilla (8).

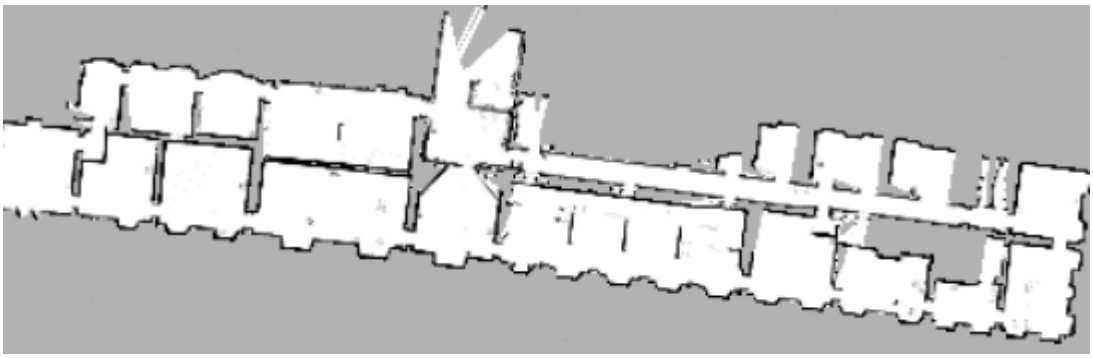

Ilustración 2.1-2. Mapa de Ocupación de Celdilla.

Para el cálculo de la probabilidad de ocupación de una celda dado la lectura de un sensor se utilizará la siguiente expresión *P(OCU|r)* siendo *OCU* la probabilidad de estar ocupada a priori y *r* la probabilidad de lo que dice el sensor es correcto. Para calcular esta probabilidad se utiliza la formulación bayesiana de la fórmula *2.1-1*:

$$P(OCU \mid r) = P(r \mid OCU) * P(OCU) \Big/ P(r \mid OCU) * P(OCU) + P(r \mid \neg OCU) P(\neg OCU)$$

2.1-1

P(OCU) y *P(¬OCU)* es la probabilidad a priori y representa lo que se conoce del entorno. *P(¬OCU)* se puede simplificar a *1-P(OCU)*. Lo mismo pasa para *P(r|¬OCU)=1-P(r|OCU)*. Conforme el robot se vaya moviendo se producirán varias lecturas y se debe actualizar el valor de la rejilla. En la primera lectura se actualiza la rejilla con la regla de Bayes descrita anteriormente. En la segunda lectura *P(OCU)* tomará el valor de *P(OCU|r)*. Este proceso se puede realizar de manera iterativa con el siguiente algoritmo:

```
Inicialmente, todas las celdillas tienen valor 0.5
    En cada instante de tiempo, tomamos las lecturas de los sensores
    Para cada lectura r
       Para cada celdilla c de la rejilla global
       Actualizar el valor de la celdilla P(OCU|r) aplicando la
siguiente fórmula:
 P(OCU|r)=P(r|OCU)P(OCU)/(P(r|OCU)P(OCU)+(1-P(r|OCU))(1-P(OCU)))
```

Las ventajas que presenta este tipo de mapas es que utilizan un algoritmo robusto, fácil de implementar y distingue entre zonas ocupadas y zonas vacías, consiguiendo una diferencia entre zonas del espacio explorado. Debido a este motivo es popular en tareas de navegación.

Las desventajas que tiene utilizar este tipo de mapas es la de cerrar bucles debido a la mala asociación de datos. Otro problema es el la dificultad que tiene la relación entre la resolución de la malla y el tiempo computacional.

2.1.4 Mapas de Características

Los mapas de características son aquellos en los que los elementos del entorno son definidos mediante puntos o líneas. La localización se lleva a cabo extrayendo las características de los datos observados y comparándolas con los que hay en el mapa. Las diferencias entre las características y los datos observados son utilizadas para calcular la posición del robot (9).

Los puntos característicos de un mapa se asumen que son perfectamente conocidos a priori y que cada uno de ellos está definido por sus correspondientes parámetros. Esta manera de definir los parámetros del mapa es muy eficiente computacionalmente, ya que no se representa el espacio libre. El inconveniente que presentan los mapas de características es que no favorece la tarea de planificación

La desventaja de estos mapas es de Asociación de Datos, ya que una incorrecta relación entre los puntos característicos del sistema y los definidos en el mapa da un aumento del error estimado, debido a una mala localización de la pose del robot. Estos mapas no son los aconsejables para entornos donde los elementos geométricos son fácilmente reconocibles.

2.2 Problema de Planificación de Cambios

En esta parte del capítulo será un estudio exhaustivo del Problema de Planificación de Cambios, dando una formulación matemática al problema y analizando las posibles soluciones, así como los métodos utilizados para crear estas soluciones.

La generación de mapas tiene como finalidad la posible navegación por el entorno de trabajo de una manera más eficiente facilitando las tareas de planificación y localización.

La definición de navegación para un robot móvil significa la acción de recorrer un camino desde una posición inicial hasta una posición final, pasando por una serie submetas. El problema de navegación se subdivide en las siguientes tareas:

- Percepción del Mundo: mediante el uso de sensores, crear un mapa del entorno en el que se encuentra el robot. (González, 1.993).

- Generación del camino: En primer lugar def ne una función continua, que interpola la secuencia de objetivos en la planificación. Posteriormente procede a la discretización de la misma con el fín de generar el camino.

- Seguimiento del camino: Efectúa el desplazamiento del robot, mediante el camino generado por los actuadores del robot. (Martínez, 1.994).

Estas subtareas de la navegación se pueden realizar por separado. El la *Ilustración 2.1-1* muestra como interaccionan estas tareas.

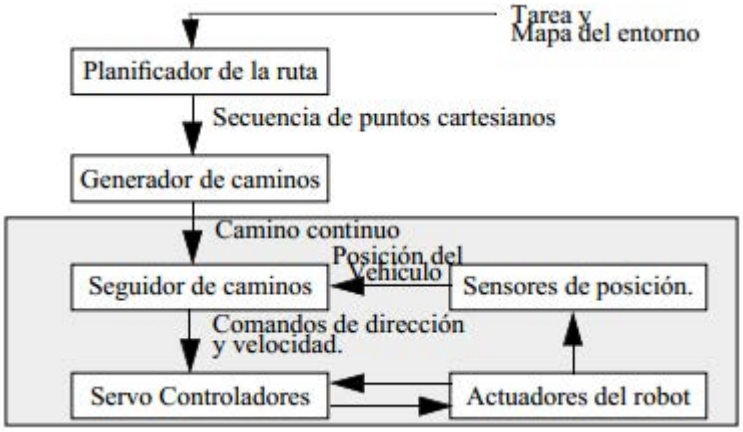

Ilustración 2.2-1. Esquema de Navegación de Robots Móviles.

2.2.1 Aproximación al Problema de Planificación de Cambios

El diseño de robots autónomos que sean capaces de realizar tareas desentraña en muchos problemas como se ha descrito anteriormente, uno de ellos es el de Planificación de Cambios. Una vaga definición de este problema sería:

- Dado un robot y un entorno con obstáculos fijos, encontrar un camino desde un punto predefinido hasta otro punto de la manera más eficiente posible.

La tarea de planificación de una ruta se subdivide en dos subtareas (Levi, 1.987):

- Planificación Global: Esta planif cación es una aproximación al camino f nal, que se va a seguir, ya que en la realización de esta acción no se consideran los detalles del entorno local al robot.

- Planificación Local: Esta tarea se encarga de evitar los obstáculos sobre la ruta global para determinar la ruta real que será seguida por el robot. El modelo del entorno local se construye mediante la fusión de la información proporcionada por los sensores externos del robot móvil. Mediante el análisis de estos datos se actualiza el modelo preliminar del entorno y se decide si se precisa replanif car la ruta local del robot.

Cabe destacar que a la hora de construir un camino o ruta para un robot móvil, no solo hay que tener en cuenta la distancia de este o la manera en la que evita los obstáculos, sino que también la velocidad a la que el robot puede circular por el trazado debido a sus características. Por eso, la tarea de Planificación debe satisfacer las siguientes restricciones:

- Restricciones debidas a las características físicas del vehículo: Se ref eren a las limitaciones impuestas por el comportamiento cinemático y dinámico del robot móvil. Ambos imponen restricciones a la velocidad máxima que puede desarrollar el vehículo según las características del camino que se desea recorrer.

- Restricciones debidas a requerimientos operacionales: En misiones donde el robot se encuentra integrado como un elemento más de un sistema, debe adaptar la velocidad de navegación según a la capacidad de cómputo de la información que desee procesar.

Por lo tanto, el desarrollo de la tarea de Planificación debe de tener como objetivo, construir una trayectoria libre de obstáculos, que conduzca al robot móvil desde una posición inicial en el plano, hasta un estado meta determinado. La trayectoria descrita debe ser admisible por las restricciones del vehículo, así como verificar ciertas restricciones operacionales.

2.2.1 Formulación Matemática del Problema de Planificación

El entorno de trabajo de un robot puede considerarse como un conjunto de configuraciones (Lozano-Pérez, 1.983) donde se puede situar al robot en un instante de tiempo. Ciertas configuraciones del entorno serán inalcanzables debido a la ocupación de obstáculos (10).

Se define una configuración *q* como un vector cuyas componentes definen el estado del robot en un instante de tiempo. Un robot es un objeto móvil al cual se le puede asignar unas coordenadas de movimiento. La localización del vehículo en un determinado instante de tiempo queda def nido por la relación existente entre el sistema de coordenadas global *Fg* en virtud del cual está def nido todo el entorno de trabajo y su sistema de coordenadas locales asociado *Fr* (11). La *Ilustración 2.2-2* muestra la configuración del robot en un problema de planificación explicado anteriormete

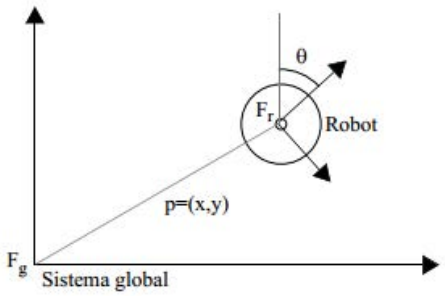

Ilustración 2.2-2. Configuración de un Robot en el Problema de Planificación.

El vector que proporciona información sobre el estado actual del robot viene dado, en principio, por dos componentes: la posición *p* y la orientación *q*. Por tanto, se puede def nir la conf guración como muestra la expresión *2.2-1*:

$$q = (p, \theta) = (x, y, \theta) \qquad 2.2\text{-}1$$

Se denomina espacio de conf guraciones *C* del robot *R* a todas las conf guraciones *q* que puede tomar el robot en su entorno de trabajo. El subconjunto de *C* ocupado por el robot *R* cuando este se encuentra en *q*, se denota por *R(q)*. Si el robot se modela de forma circular con radio *r, R(q)* se def ne como muestra la expresión *2.2-2*:

$$R(q) = \{q_i \in C / \|q, q_i\| \leq \rho\} \quad 2.2\text{-}2.$$

En caso de un robot puntual, ρ es nulo con lo cual se cumple la expresión *2.2-3*.

$$R(q) = \{q\} \quad 2.2\text{-}3$$

En el espacio de trabajo, donde el robot realizará su tarea, se encuentran distribuidos una serie de obstáculos def nidos como un conjunto de objetos rígidos *B* (expresión *2.2-4)* y que se encuentran distribuidos por el espacio de conf guraciones *C*.

$$B = \{b_1, b_2, \ldots, b_q\} \quad 2.2\text{-}4$$

El conjunto de conf guraciones del espacio *C* ocupadas por un obstáculo se def ne por $b_i(q)$, de tal forma que el subconjunto de conf guraciones de *C*, que especif ca el espacio libre de obstáculos viene dado por la expresión *2.2-5:*

$$C_l = \{q \in C / R(q) \cap \left(\bigcup_{i=1}^{q} b_i(q) \right) = \varnothing\} \quad 2.2\text{-}5$$

El problema de la planif cación, tal y como se ha def nido, queda transformado en la búsqueda de una sucesión de posturas *q* tal que la primera de ellas sea la postura actual del robot q_a y la última de esta sucesión la postura objetivo q_f. Todas las posturas de la serie deben pertenecer al subconjunto C_l def nido en la expresión definido en la expresión *2.2-5*. Es decir, una ruta Q_r que conecta la postura inicial q_a con la f nal q_f es la expresión *2.2-6*:

$$Q_r = \{q_a, \ldots, q_f / q_i \in C_l\} \quad 2.2\text{-}6$$

La especif cación de este conjunto Q_r, implica la construcción de una función ruta def nida de la expresión *2.2-7:*

$$\tau : [0, 1] \rightarrow C_l \quad 2.2\text{-}7$$

Tal que:

$$\tau(0) = q_a \qquad \tau(1) = q_f \quad 2.2\text{-}8$$

Además de la restricción mostrada en anterior la expresión, se le exige a la función τ, teniendo en cuenta la suposición de robot un omnidireccional, la continuidad. Este concepto se ref eja en la expresión *2.2-9*:

$$\lim_{s \to s_0} \|\tau(s), \tau(s_0)\| = 0 \qquad 2.2\text{-}9$$

2.2.1 Métodos Clásicos de Planificación

Estos métodos se fundamenta en la construcción de un grafo en su parte inicial sobre el espacio libre, para posteriormente emplear un algoritmo de búsqueda en grafos como *A**, este tipo de algoritmos encuentra el camino óptimo según cierta función de coste.

2.1.3.1 Planificación basada en Grafos de Visibilidad

Los grafos de visibilidad (Nilsson, 1.969) proporcionan un enfoque geométrico útil para resolver el problema de la planif cación. Para la generación del grafo este método introduce el concepto de *visibilidad*, según el cual def ne dos puntos del entorno como visibles si y solo si se pueden unir mediante un segmento rectilíneo que no intersecte ningún obstáculo (si dicho segmento resulta tangencial a algún obstáculo se consideran los puntos afectados como visibles) (12).

La *Ilustración 2.2-3* muestra un grafo resultante de este método.

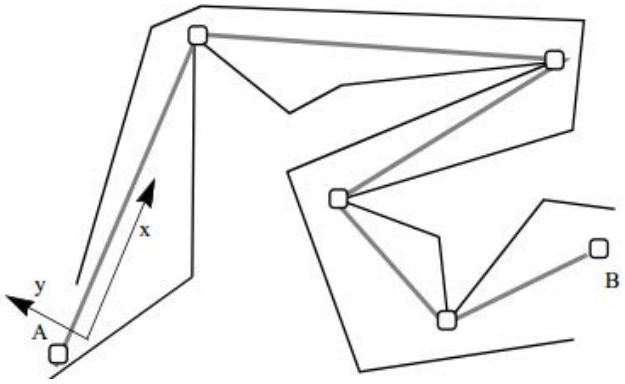

Ilustración 2.2-3. Grafos de Visibilidad.

Este tipo de métodos está muy extendió debido a su bajo coste computacional. Sin embargo, utilizar como nodos los vértices de los obstáculos implica que no son inmediatamente aplicables en la práctica, ya que un robot móvil real no consiste en un punto La ruta creada por este modelo se le denomina ruta de semi-obstáculo, debido al problema explicado anteriormente (12).

2.1.3.2 Planificación basada en Diagramas de Voronoi

Al contrario que los grafos de visibilidad los diagramas de Voronoi sitúan el camino los más alejado posible de los obstáculos, eliminado el problema de la semi-ruta libre de obstáculos (13).

Los diagramas de Voronoi están definidos como una proyección del espacio libre del entorno en una red de curvas unidimensionales yacientes en dicho espacio libre.

Formalmente se def nen como una retracción (Janich, 1.984) con preservación de la continuidad. Si el conjunto C_l def ne las posiciones libres de obstáculos de un entorno, la función retracción *RT* construye un subconjunto C_v continuo de C_l de la forma que muestra la expresión *2.4-10*.

$$RT(q):\mathrm{C}_l \rightarrow C_v / C_v \subset C_l \quad 2.2\text{-}10$$

De esta forma, existe un camino desde una conf guración inicial q_a hasta otra f nal q_f, ambas libres de obstáculos, si y solo si existe una curva continua desde $RT(q_a)$ hasta $RT(q_f)$.
Como se ha comentado anteriormente la idea principal es de alejarse lo máximo posible de obstáculos., luego el diagrama de Voronoi resulta el lugar geométrico de las configuraciones de las distancias de los objetos que se encuentra más próximos en el entorno. Los segmentos que conforma el diagrama de Voronoi son rectilíneos y parabólicos. El tipo de segmento utilizado en cada situación dependerá de la distancia de los objetos que se encuentren enfrentados entre sí. De esta forma, el lugar geométrico de las conf guraciones que se hallan a igual distancia de dos aristas de dos obstáculos diferentes, es una línea recta, mientras que en el caso de tratarse de un vértice y una arista resulta una parábola como muestra la *Ilustración 2.2-4* (14).

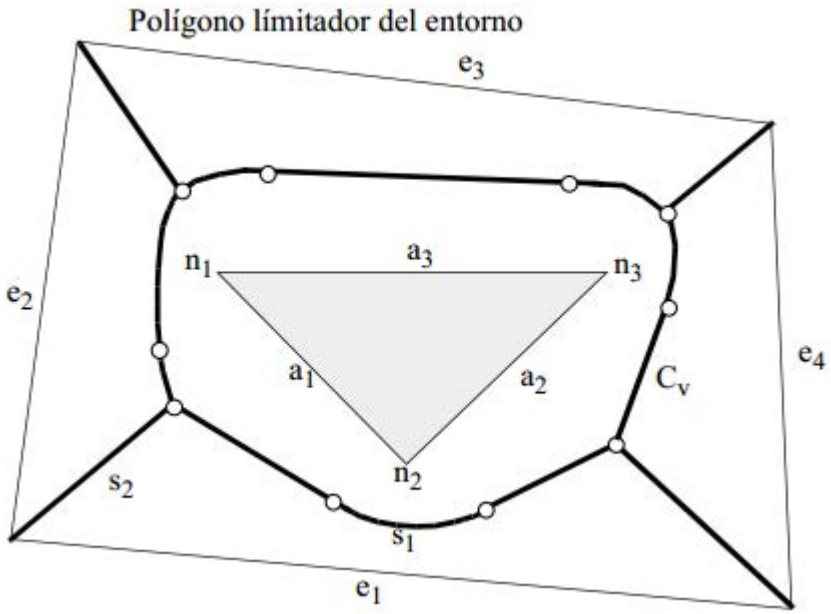

Ilustración 2.2-4. Diagrama de Voronoi.

El entorno está delimitado por $\{e_1,e_2,e_3,e_4\}$ de la *Ilustración 2.4-1* y el obstáculo es el triángulo central. La retracción del espacio libre en una red continua de curvas es el diagrama de Voronoi C_v, representado mediante las líneas de trazo grueso. Esta línea se define con la función *RT(q)* que es el primer corte con C_v de la línea que une *p* con *q* como muestra la *Ilustración 2.2-5.*

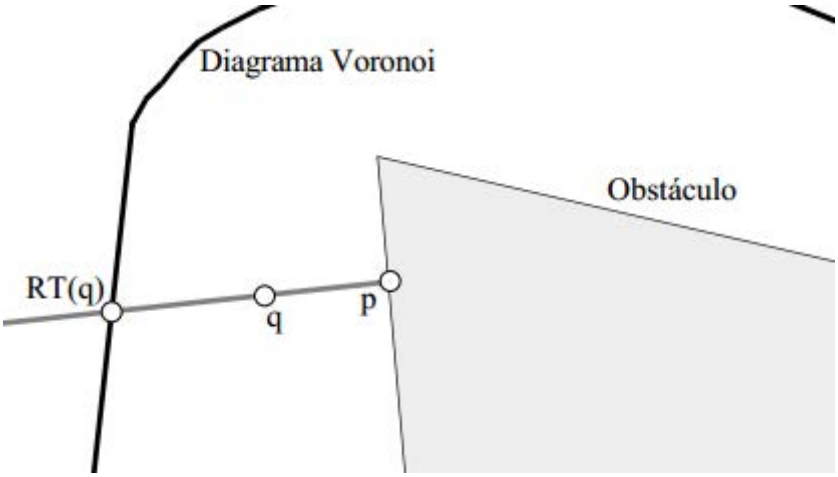

Ilustración 2.2-5. Construcción Diagrama de Voronoi.

El algoritmo de planif cación, consiste en encontrar la secuencia de segmentos s_i del diagrama de Voronoi tal que conecten $RT(q_a)$ con $RT(q_f)$.
El pseudocódigo del este algoritmo es el siguiente:

```
1. Calcular el diagrama de Voronoi.
2. Calcular RT (qa) y RT(qf).
3. Encontrar la secuencia de segmentos {s1,...,sp } tal que RT(qa)
pertenece a s1 y RT(qf) pertenece a sp.
4. Si se encuentra dicha secuencia, devolver la ruta. Si no
indicar condición de error
```

2.1.3.3 Planificación basada en el Modelado del Espacio Libre

Esta metodología se aplica en entornos donde la los obstáculos tienen forma poligonal. Esta técnica está basada en los llamados cilindros rectilíneos generalizados (CRG). Esta práctica es parecida a los diagramas de Voronoi, ya que con los cilindros (CRG) lo que se pretende es que el robot navegue los más alejado posible de los obstáculos.

La construcción de los CRG se realiza a partir de las aristas de los obstáculos. Para que un par de aristas ${}^1a_i$ y ${}^2a_j$ pertenecientes a los obstáculos b_1 y b_2 respectivamente puedan formar un cilindro generalizado, deben cumplir las siguientes condiciones:

- La arista ${}^1a_i$ debe estar contenida en una recta que divide al plano en dos regiones. La arista ${}^2a_j$ debe yacer por completo en la región opuesta en la que se encuentra situada b_1. Este criterio es simétrico.

- El producto escalar de los vectores normales con dirección hacia el exterior del obstáculo que contiene cada arista debe resultar negativo.

Si se cumple estas condiciones se puede crear un CRG, véase la *Ilustración 2.2-6*.

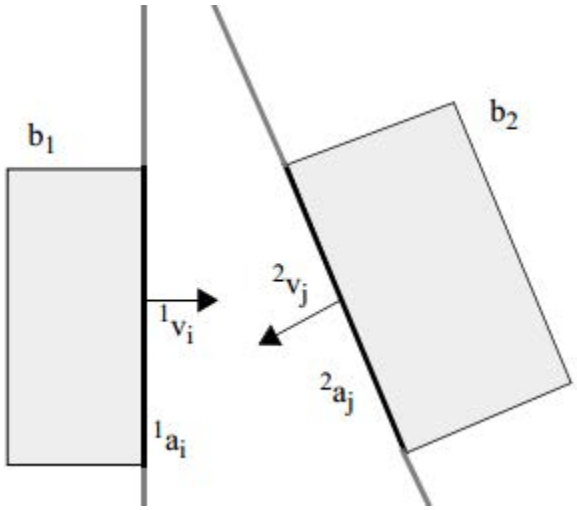

Ilustración 2.2-6. Creación de un CRG.

Una vez detectado el CRG solo queda construirlo, este proceso aparece en la *Ilustración 2.2-7*.

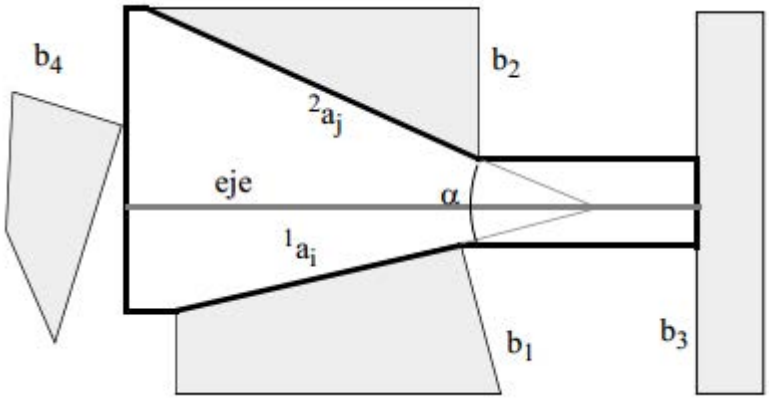

Ilustración 2.2-7. Construcción Mapa con CRG.

El primer paso es el cálculo del eje del CRG, el cual se def ne como la bisectriz del ángulo α formado por el corte de las rectas que contienen las aristas ${}^{1}a_i$ y ${}^{2}a_j$ que cumplen las dos condiciones expuestas más arriba. Por ambos lados de dichas aristas se construyen segmentos rectilíneos paralelos al eje, con origen en los vértices de las aristas implicadas y con extremo señalado por la proyección del primer obstáculo que corta el eje.

El robot navegará por el eje del cilindro en el cual se encuentra anotadas todos los rangos de orientaciones admisibles. El paso de un CRG a otro se produce siempre y cuando sus ejes intersecten y la intersección del rango de orientaciones admisibles en el punto de corte de ambos ejes no sea nulo.

2.3 Path Planning

El objetivo que aborda el problema de Path Plannig es el de obtener la ruta más óptima entre un punto y otro. La cuestión es ¿Cuál es la ruta más óptima? Esto dependerá de las restricciones que se consideren más importantes, estas restricciones pueden ser:

- Tiempo: El tiempo que conlleva el desplazarse de un punto a otro.
- Distancia: La distancia que hay entre puntos.

Para abordar este problema es necesario marca en un mapa una serie de Waypoints. Estos waypoin se pondrán en los sitos del mapa que puede alcanzar el robot. Una vez que se tienen marcados estos puntos se realiza una conexión entre los adyacentes, creando se así un grafo de adyacencia. Para obtener la ruta más óptima solo que da aplicar un algoritmo búsqueda.

En el siguiente apartado se realiza un estudio de la descomposición en celdas de los mapas para la creación de Waypoints.

2.3.1 Path Planning basado en la Descomposición en Celdas

Este tipo de métodos se fundamental en las descomposición del espacio libre (Thorpe, 1,984). La búsqueda de una ruta desde una postura inicial q_a hasta otra f nal q_f, consiste en encontrar una sucesión de celdas que no presente discontinuidades, tal que la primera de ellas contenga q_a y la última a q_f.

Este modelo plantea dos problemas fundamentales que son la división del espacio libre en celdas y la creación de un grafo de conectividad. El primero de estos problemas consiste en crear diferentes celdas con una configuración geométrica que resulte fácil de calcular el camino entre dos configuraciones distintas pertenecientes a la celdas y el proceso de comprobación de si dos celdas son adyacentes debe de ser lo más simple posible. Además no debe haber solapamiento entre celdas y la unión de todas ellas debe constituir el espacio libre modelado.

El grafo de conectividad es no dirigido y su construcción se basa en la asignación de un nodo por cada celda construida en el paso anterior, siendo los arcos de este grafo las uniones con los nodos que representan las celdas adyacentes como muestra la *Ilustración 2.3-1*.

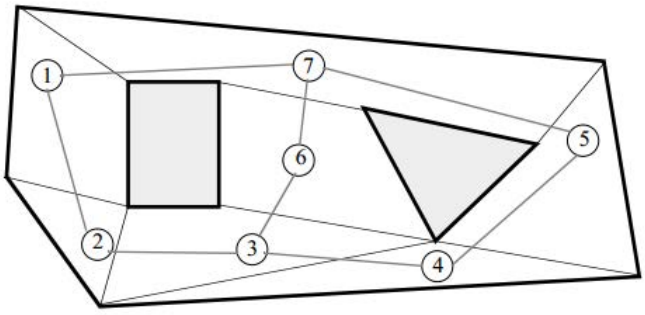

Ilustración 2.3-1. . Descomposición en Celdas.

Una vez realizado el grafo de adyacencia solo cabe aplicar un algoritmo de búsqueda que dado una posición inicial q_a y una posición final q_f encuentre el camino de menor peso.

Los distintos métodos basados en este principio, se distinguen por la forma en la cual realizan la descomposición en celdas. El método más sencillo de descomposición del espacio libre del entorno en celdas se llama *descomposición trapezoidal* (Latombe, 1,991). Este método está basado en la construcción de segmentos rectilíneos paralelos al eje *Y* del sistema global F_g, a partir de los vértices de cada uno de los elementos del entorno. El final del segmento queda delimitado por el primer corte de la línea con un elemento del entorno. Este método se muestra en la *Ilustración 2.3-2*.

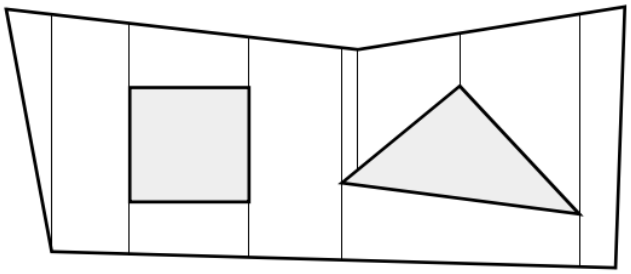

Ilustración 2.3-2. Descomposición en Celdas (descomposición trapezoidal).

Por último, cabe destacar que este tipo de modelos pueden realizar una descomposición de un espacio en 3D solo con aplicar otra dimensión más a la descomposición del espacio libre.

2.4 Construcción de los mapas y posicionamiento de robots

En este apartado se verá la base matemática de los problemas de localización y modelado simultaneo que ha surgido en este proyecto. Se realizará la descripción del problema SLAM (Simultaneous localization and mapping)

2.4.1 SLAM

El problema de SLAM empezó a ser tratado en profundidad cuando los problemas de modelado y localización empezaron a ser muy complicados y en ciertos casos imposibles de resolver si se hacían por separado.

Los estudios de Smith y Cheeseman y Durrant-Whyte establecieron las bases estadísticas para describir relaciones espaciales entre los diferentes elementos del entorno y el manejo de incertidumbres con las medias tomadas del entorno. El artículo de Smith, Self y Cheeseman (15) plasmo la idea, de que el robot a medida que realiza un desplazamiento por un entorno comete cierto error en sus medidas.

A pesar de los avances tecnológicos de hoy en día, los sensores de un robot cometen cierto error a la hora de tomar medidas de los objetos de un entorno. Para subsanar estos problemas se formularon varias soluciones. Las mejores soluciones que se han obtenido a la hora de tratar el problema de SLAM son aquellas basadas en técnicas probabilísticas, estas técnicas hacen referencia a todas las medidas de incertidumbre que pueden surgir en el proceso. Este tipo de algoritmos, tienen su base en el teorema de Bayes, que relaciona entre sí las probabilidades marginal y condicional de dos variables aleatorias.

2.1.2.1 Formulación Matemática del problema de SLAM

Se considera un robot moviéndose en un entorno desconocido tomando referencia un número de marcas en un tiempo *k*, para ello se realiza las siguientes definiciones:

- $\mathbf{x}_k$: Es el vector de estado que describe la orientación y localización del robot.
- $\mathbf{U}_k$: es el vector de control, aplicando en el tiempo *k-1*, para conducir al robot al estado x_k en el tiempo *k*.
- $\mathbf{m}_i$: es un vector que describe la posición de la *i-ésima* marca cuya verdadera localización es invariante en el tiempo.
- $\mathbf{Z}_{ik}$: es una observación tomada desde el robot de la localización de la *i-ésima* marca en tiempo *k*.

Con estas cantidades define los siguientes conjuntos:

- $\mathbf{X}_{0:k} = (x_0, x_1, \ldots, x_k) = (X_{0:k-1}, x_k)$. Historial de localización del robot.
- $\mathbf{U}_{0:k} = (u_0,u_1,\ldots, u_k) = (U_{0:k-1},u_k)$: Historial de salidas de control del robot.
- $\mathbf{M} = (m_1, m_2,\ldots, m_n)$: Conjunto de marcas.
- $\mathbf{Z}_{0:k}=(z_1,z_2,\ldots,z_k) = (Z_{0:k-1},z_k)$:Conjunto de todas las marcas observadas.

El problema de SLAM requiere de la distribución de probabilidad de la expresión *2.4-1*:

$$P(x_k, m \mid Z_{0:k}, U_{0:k,x_0}) \quad 2.4\text{-}1$$

Esta distribución probabilística se calcula para todos los tiempos *k*. Esta distribución describe la densidad posterior de la localización de la marca y el estado del robot en el tiempo *k*, dada las observaciones almacenadas y las distribuciones de controles, hasta incluir un tiempo *k* junto con el estado inicial del robot. Con una estimación de la distribución $P(x_{k-1}, m \mid Z_{0:k-1}, U_{0:k-1})$ en el tiempo *k-1*, la unión posterior, siguiendo un control $\mathbf{u}_k$ y una observación $\mathbf{z}_k$ es computarizada usando el Teorema de Bayes. Este cálculo requiere que un modelo de transición de estado y un modelo de observación sean definidos describiendo las salidas de control y de observación respectivamente.

Un modelo de observación describe la probabilidad de hacer una observación $\mathbf{z}_k$ cuando la ubicación del robot y la marca son conocidas y es descrita como muestra la expresión *2.4-2:*

$$P(z_k \mid \mathrm{x}_k, m) \quad 2.4\text{-}2$$

Una vez que se asume la localización del robot y el mapa es definido, las observaciones son condicionalmente independientes dado el mapa y el estado actual del robot.

El modelo de movimiento del robot puede ser descrito en términos de la distribución de probabilidad sobre la transición de la expresión *2.6-3*:

$$P(x_k \mid \mathrm{x}_{k-1}, u_k) \quad 2.4\text{-}3$$

De este modo, el estado de transición es como un proceso markoviano[3] en el cual el estado próximo a $\mathbf{x}_k$ depende únicamente del estado inmediatamente precedente a $\mathbf{x}_{k-1}$ y el control aplicando $\mathbf{u}_k$ y es independiente tanto de las observaciones como del mapa.

El algoritmo SLAM es implementado en un estándar recursivo de dos pasos, el primero de ellos es la propagación (actualización del tiempo) y corrección (actualización de medida) en la forma:

Actualización de tiempo se produce dada la expresión *2.4-4*.

$$P(x_k, m \mid \mathrm{Z}_{0:k}, \mathrm{U}_{0:k, x_0}) = \int P(x_k \mid \mathrm{x}_{k-1}, u_k) P(x_{k-1}, m \mid \mathrm{Z}_{0:k-1}, U_{0:k-1}, x_0) dx_{k-1} \quad 2.4\text{-}4$$

Actualización de medidas se produce dada la expresión *2.4-5*.

$$P(x_k, m \mid \mathrm{Z}_{0:k}, \mathrm{U}_{0:k, x_0}) = \frac{P(z_k \mid \mathrm{x}_k, m) P(x_k, m \mid \mathrm{Z}_{0:k-1}, U_{0:k}, x_0)}{P(z_k \mid \mathrm{Z}_{0:k-1}, U_{0:k})} \quad 2.4\text{-}5$$

Las ecuaciones de Actualización de Tiempo y Actualización de Medidas dan un procedimiento recursivo para calcular la unión posterior. $P(x_k, m \mid \mathrm{Z}_{0:k}, \mathrm{U}_{0:k, x_0})$ Para el estado del robot x_k y el mapa *m* en el tiempo *k* basado en todas las observaciones $\mathrm{Z}_{0:k}$ y todas las salidas de control $\mathrm{U}_{0:k}$ hasta incluir en el tiempo *k*. La función recursiva depende del modelo del robot $P(x_k \mid \mathrm{x}_{k-1}, u_k)$ y el modelo de observación $P(z_k \mid \mathrm{x}_k, m)$.

Hay que anotar que el problema de la construcción del mapa puede ser formulado calculando la densidad condicional $P(m \mid \mathrm{X}_{0:k}, Z_{0:k}, U_k)$. Esto asume que la ubicación del robot x_k es conocida, o por lo menos determinada en cada instante, sujeto al conocimiento previo de la posición inicial. Un mapa *m* es construido a través de combinar observaciones de diferentes posiciones. Prácticamente el problema de localización puede ser formulado calculando la distribución de probabilidades $P(m \mid \mathrm{X}_{0:k}, Z_{0:k}, U_k, m)$. Esto asume que la localización de las marcas es conocida con certeza y que el objetivo es calcular y estimar la localización del robot respecto a estas marcas.

[3] Proceso Markoviano. Un proceso Markoviano es un proceso estocástico en el que la probabilidad de que ocurra un evento de pende directamente del evento anterior.

2.1.2.2 Formulación Bayesiana del problema de SLAM

Los algoritmos que construyen mapas modelan la posición del robot y su entorno de manera probabilística y utilizan la inferencia para realizar una transformación en las medidas, obtenidas por el robot, aun mapa probabilístico. Esta formulación es la más extendida, ya que la incertidumbre presente en los sensores y el movimiento del robot hace que sea el planteamiento más adecuado.

Un robot móvil explorando un ambiente desconocido deberá moverse a través del entorno mediante las medidas que tomen sus sensores. En este caso se distinguirán dos tipos de datos, las medidas odométricas y las medidas recogidas por los sensores de percepción. Se asume pues que estas medidas llegan de forma secuencial alternativa en el tiempo de la siguiente manera:

$\mathbf{u}_1, \mathbf{z}_1, \mathbf{u}_2, \mathbf{z}_2, \ldots, \mathbf{u}_t, \mathbf{z}_t$

Siendo **u** el desplazamiento relativo a la posición anterior del robot y **z** una medida sensorial tomada por un sensor respecto al robot y un objeto del entorno. Donde *t* representa el tiempo.

El estado del sistema $\mathbf{x}_t$ en el instante *t* está dado por la posición del robot $\mathbf{s}_t$ y la posición de cada uno de los objetos del mapa m_t.

$$x_t \equiv \begin{bmatrix} s_t \\ m_t \end{bmatrix}$$

La densidad de probabilidad del estado $\mathbf{x}_t$, condicionado al conjunto de todos los datos almacenados hasta el instante *t*, se representa como muestra la expresión *2.6-1*.

$$p(x_t \mid \mathrm{z}^{\mathrm{t}}, u^t) = p(s_t, m_t \mid \mathrm{z}^{\mathrm{t}}, u^t) = \eta p(z_t \mid \mathrm{s}_{\mathrm{t}}, m_t, z^{t-1}, u^t)\, p(s_t, m_t \mid \mathrm{z}^{\mathrm{t\text{-}1}}, u^t) \quad 2.4\text{-}1$$

η es un factor de normalización de la función de probabilidad. Esta función de probabilidad, representa cualquier solución probabilística al problema SLAM.

Aplicando la hipótesis de Markov, en el cual, el único estado que existe es le representado por $\mathbf{s}_t$ y $\mathbf{m}_t$ de la ecuación anterior, se puede escribir como la expresión *2.6-2*:

$$p(s_t, m_t \mid \mathrm{z}^{\mathrm{t}}, u^t) = \eta p(z_t \mid \mathrm{s}_{\mathrm{t}}, m_t)\, p(s_t, m_t \mid \mathrm{z}^{\mathrm{t\text{-}1}}, u^t) \quad 2.4\text{-}2$$

Utilizando la ley de probabilidad total se puede expresar el segundo factor de la derecha de la expresión *2.4-3*:

$$p(s_t, m_t \mid \mathrm{z}^{\mathrm{t\text{-}1}}, u^t) = \iint p(s_t, m_t \mid \mathrm{z}^{\mathrm{t\text{-}1}}, u^t, s_{t-1}, m_{t-1}) p(s_{t-1}, m_{t-1} \mid \mathrm{z}^{\mathrm{t\text{-}1}}, u^t) ds_{t-1} dm_{t-1} \quad 2.4\text{-}3$$

Haciendo nuevamente la hipótesis de Markov y asumiendo que el movimiento del robot es totalmente independiente del mapa construido, se tiene la expresión *2.4-4*:

$$p(s_t, m_t \mid z^t, u^t) = \int p(s_t \mid u_t, s_{t-1}) p(s_{t-1}, m \mid z^{t-1}, u^{t-1}) ds_{t-1} \quad 2.4\text{-}4$$

Con lo que quede se reduce a la expresión *2.4-5*.

$$p(s_t, m_t \mid z^t, u^t) = \eta p(z_t \mid s_t, m_t) \int p(s_t \mid u_t, s_{t-1}) p(s_{t-1}, m \mid z^{t-1}, u^{t-1}) ds_{t-1} \quad 2.4\text{-}5$$

Por lo que la solución al problema de SLAM se puede hacer de forma recursiva, necesitando en cada ínstate de tiempo los datos sensoriales obtenidos para ese instante de tiempo con su probabilidad y la función de probabilidad del estado en el instante anterior (16). La ecuación anterior se conoce como filtro de Bayes para solucionar el problema de SLAM. Para evaluar la expresión anterior se requiere de dos funciones de probabilidad generativas. Estas funciones corresponden al sistema odométrico y a de percepción externa. Por lo general, estas dos funciones son invariantes en el tiempo, con lo que el modelo de medida se puede expresar como $p(z_t \mid s_t, m_t)$ y el modelo de movimiento por $p(s_t \mid u_t, s_{t-1})$.

El problema principal es que la expresión *2.4-5* no puede implementarse ni resolverse en su forma general, por lo que es necesaria la realización de simplificaciones, suposiciones o hipótesis añadidas a la solución.

2.1.2.3 Filtro de Kalman

Una de las funciones más extendidas de la ecuación *2.4.-5* del filtro de Bayes es la basa en el filtro de Kalman. El filtro de Kalman es un estimador recursivo lineal de mínimos cuadrados. Este método muy utilizado en el área de la robótica, la estimación con incertidumbre y fusión sensorial. La aplicación para sistemas no lineales se hace mediante la extensión del filtro de Kalman al linealizar las ecuaciones, cuyo nombre es Filtro Extendido de Kalman (EKF) (17).

El EKF se deriva de suposición de que la probabilidad del estado $p(s_t, m_t \mid z_t, u_t)$ se puede modelar por una distribución de probabilidad unimodal Gaussiana multidimensional, caracterizada por su valor esperado y su matriz de varianzas covarianzas.

$$x_t = s_t\, m_t \Leftrightarrow x(k)$$
$$x(k) \sim \mathcal{N}(\dot{x}(k \mid k), P(k \mid k))$$

Dónde:

$$\dot{x}(k \mid k) = E[x(k)]$$
$$P(k \mid k) = E[((x(k) - \dot{x}(k \mid k))(x(k) - \dot{x}(k \mid k))^T]$$

Por la propia naturaleza del EKF, esta solución requiere el uso de un mapa geométrico, basado en elementos discretos con determinadas características geométricas para poder ser parametrizado en el vector de estado a estimar por el algoritmo.

La solución al problema de SLAM con EKF tiene como principal ventaja que es capaz de mantener la estimación completa a posteriori explícitamente de forma incremental, a costa de suponer que tanto el ruido del sistema odométrico como del sistema sensorial pueden ser aproximados por una distribución normal.

El EKF requiere para el cálculo de su matriz de covarianza que tanto el modelo de movimiento como el de medida, los cuales por lo general son lineales, sean linealizados. Generalmente, se supone que esta linealización es aceptable y que los errores cometidos son pequeños, lo que es asumido por la mayoría de las implementaciones existentes.

2.1.2.4 Método de Monte Carlo

En el caso de las técnicas de muestreo, como pueden ser los algoritmos de aproximación, se basan en asumir aleatoriamente la existencia de valores para algunos nodos y luego usar estos valores para inferir en la cantidad de otros nodos. Luego, se mantiene las estadísticas de los valores que estos nodos toman, y finalmente estas estadísticas dan la respuesta. A estos métodos se les llama técnicas de Monte Carlo (10).

El método de Monte Carlo lo que intenta es encontrar el valor esperado de alguna función *f(x)* con respecto a una distribución de probabilidad *p(x)* donde *x* puede estar formada por variables discretas, continuas, o alguna combinación de ambas. En el caso de variables continuas se quiere calcular el valor esperado en la expresión *2.4-6*:

$$f = \int f(x)p(x)dx \quad 2.4\text{-}6$$

La idea principal de este algoritmo se basa en representar la función de densidad de probabilidad de la posición del robot mediante un conjunto finito de partículas, $\{x^{(i)}, P(x^{(i)})\}$ $i = \{1,...,N\}$, donde x^i representa la ubicación (x, y, Θ) del robot y $P(x^i)$ su probabilidad asociada, calculadas mediante un modelo de observación. Se asume que $\sum_{i=0}^{N} P(x^i) = 0$.

Inicialmente, el conjunto de muestras se distribuye uniformemente de forma aleatoria por todo el espacio de posición posible. Las muestras se generan en cada instante, $M_t = \{x_{(t)}^{(1)}, x_{(t)}^{(2)}, ..., x_{(t)}^{(n)}\}$ a partir de las muestras del instante anterior, $M_t = \{x_{(t-1)}^{(1)}, x_{(t-1)}^{(2)}, ..., x_{(t-1)}^{(n)}\}$ junto a las medidas del entorno z_t y las acciones ejecutadas por el robot, $u_{(t=1)}$ (desplazamiento en x, y, Θ) siguiendo tres pasos recursivamente:

Predicción:

Se desplazan las muestras según la acción ejecutada y se estima la nueva posición de las muestras. Sea u_{t-1} la acción ejecutada por el robot en el instante *t-1*. Para cada muestra $x_{t-1} \in M_{t-1}$ se predice su nuevo estado en el instante *t*, añadiendo un cierto error de movimiento, como muestra la expresión *2.4-7*;

$$x_t^{(i*)} = x_{t-1}^{(i)} + u_{t-1} + (N(0, \sigma_x), N(0, \sigma_y), N(0, \sigma_\Theta))^T \quad 2.4\text{-}7$$

Con $i = \{1, ..., N\}$. De esta forma se construye un nuevo conjunto de *N* muestras en el instante *t*.

Actualizaciones de observaciones:

Se calculan las probabilidades a posteriori de cada muestra, dada la última observación sensorial. Sea z_t la observación realizada por el robot en el instante actual *t*. Para cada muestra $x_t^{(i*)} \in M_t^*$ se actualiza su probabilidad asociada, $P(x_t^{(i*)})$, según la verosimilitud de que la observación z_t haya observada desde el estado $x_t^{(i*)}$, como muestra la expresión *2.4-8*.

$$P(x_t^{(i*)}) = P(x_t^{(i*)} \mid z_t),\ i = \{1, ..., N\} \qquad 2.4\text{-}8$$

Remuestreo:

Se genera un nuevo conjunto de muestras con distribución proporcional a la verosimilitud de las muestras anteriores. Construir un nuevo conjunto de *N* muestras, M_t, re muestreando con sustitución el conjunto M_t^*, de forma que escoja cada muestra $x_t^{(i*)}$ con probabilidad proporcional a la verosimilitud de la misma $P(x_t^{(i*)})$, como muestra la expresión *2.4-9*;.

$$(x_t^{(i)}, P(x_t^{(i)})) \longleftarrow \textit{Escoger muetras de } M_t^* \qquad 2.4\text{-}9$$

Con $i = \{1,..., N\}$. Normalizar las probabilidades $P(x_t^{(i*)})$ de las muestras de M_t de forma que $\sum_{i=0}^{N} P(x^i) = 1$ Para ello se tiene la expresión *2.4-10*:

$$P(x_t^{(i)}) \longleftarrow \frac{P(x_t^{(i)})}{\sum_{j=1}^{N} P(x^i)},\ i = \{1, ..., N\} \qquad 2.4\text{-}10$$

Siguiendo esta forma de proceder, las muestras se van concentrando alrededor de las posiciones más probables del robot, hasta que finalmente se produce la convergencia de la población de muestras en un área reducida alrededor de la posición real del sistema móvil.

2.1.2.5 Filtro de Partículas

Es un método empleado para localizar un sistema que cambia a lo largo del tiempo. Se podría decir que es un método de Monte Carlo secuencial, se utiliza en visión artificial para el seguimiento de objetos en secuencias de imágenes. Este algoritmo fue creado por N. Gordon, D. Salmond y A. Smith en 1993 (18).

El filtro de partículas realiza dos fases, que son predicción y la actualización. En este aspecto se asemeja mucho a los filtros de Kalman. Para resolver el problema de SLAM, cada partícula del filtro de partículas representa las posibles trayectorias que puede tomar el robot en el mapa.

La idea principal es estimar la distribución $p\ (S_{1:t} \mid z_{1:t}\ ;\ U_{0:t})$ sobre las trayectorias potenciales $s_{1:t}$ de un robot dadas las observaciones $z_{1:t}$ y sus mediciones de odontometría $U_{0:t}$ y utilizar la

distribución posterior para calcular una distribución posterior de mapas y trayectorias, como muestra la expresión *2.4-10*.

$$p(s_{1:t}; m \mid z_{1:t}; u_{0:t}) = p(m \mid s_{1:t}; z_{1:t}) p(s_{1:t} \mid z_{1:t}; u_{0:t}) \quad 2.4\text{-}11$$

Esto se realiza de manera eficiente ya que la distribución posterior sobre mapas $p(m \mid s_{1:t}; z_{1:t})$ puede calcularse analíticamente, dado por conocidos $s_{1:t}$ y $z_{1:t}$. Y puede hacerse esto gracias a que las trayectorias y los mapas son condicionalmente independientes. Cada mapa es construido dadas las observaciones $z_{1:t}$ y las trayectorias $s_{1:t}$ representadas por la partícula correspondiente.

La implementación de este método que se utiliza en este proyecto pertenece al proyecto OpenSlam. que se encuentra en los repositorios de ROS. Los autores del código son Giorgio Grisetti, Cyrill Stachniss, Wolfram Burgard, el repositorio de ROS es obra de Brian Gerkey (19).

2.5 Marcos de Referencia y Cuaterniones

Un marco de referencia es una conjunto de arreglos que se utiliza para definir magnitudes físicas como puede ser la posición o la orientación. En este sistema, un marco de referencia será el conjunto de coordenadas ortogonales del robot o de algún punto del entorno. En los sistemas robóticos, existen varios marcos de referencia como puede ser un sistema de coordenadas asociado a un mapa o el sistema asociado a la base móvil del robot.

Los marcos de referencia se relacionan a través de relaciones de tipo "padre-hijo" y queda descrito con un cuaternion que describe las rotaciones y un vector que define las translaciones en los ejes x, y, z.

Los cuaterniones son sistemas numéricos hipercomplejos, que aplicados en mecánica tridimensional, sirven para representar sistemas en el espacio.

Los cuaterniones pertenecen a un espacio vectorial de cuatro dimensiones sobre los números reales, hecho sobre la base de 3 vectores unitarios $\hat{\imath}, \hat{\jmath}, \hat{h}$ tal que $\hat{\imath}^2 = \hat{\jmath}^2 = \hat{h}^2 = \hat{\jmath}\hat{\imath} = -1$ luego dado esta definición, un cuaternion sigue la forma $a + b\,\hat{\imath} + c$ de $\hat{h}$.

En ROS Anexo III corresponde a un vector de dimensiones $q = [x; y; z; w]$. Un cuaternion representa una rotación α entorno al eje definido entre $[0; 0; 0]$ y $[x; y; z]$ de la siguiente manera:

$X = x * \sin(0.5 * \alpha)$

$Y = y * \sin(0.5 * \alpha)$

$Z = z * \sin(0.5 * \alpha)$

$W = \cos a(0.5 * \alpha)$

Si el cuaternion no se encuentra normalizado habría que normalizarlo. Este tipo de rotaciones se utilizará para representar las rotaciones y traslaciones en los marcos de referencia de este proyecto.

2.6 Modelado del Conocimiento con PDDL

Cuando se intenta utilizar técnicas de planif cación para resolver problemas del mundo real, aparecen de manera natural dominios ricos en conocimiento. Allen Newell (Newell, 1982) introdujo el concepto *Modelo en el Nivel de Conocimiento* para plantear la distinción entre las actividades de análisis y modelado de elementos de conocimiento (Nivel de Conocimiento) con las actividades de representación computacional de dichos elementos (Nivel Simbólico). Esta separación a hace posible determinar qué es lo que el sistema hace, y diferenciarlo de cómo realmente lo hace. En la actualidad, se considera que la construcción de un Sistema Basado en el Conocimiento (SBC) es una actividad de modelado orientada a construir un modelo computacional con una capacidad para resolver problemas comparables a la de un experto humano en el mismo dominio de la aplicación.

En planif cación independiente del dominio, siempre se distingue entre conocimiento del dominio y conocimiento del problema. El conocimiento del dominio comprende la información común a cualquier problema, es decir, def ne un marco común y delimita el tipo de problemas que se puede def nir. El conocimiento del problema completa la información del dominio def niendo una situación específ ca, en la que se persigue alcanzar ciertos objetivos concretos.

Un planificador es un sistema encargado de realizar un proceso metódico para obtener un objetivo. El objetivo es una situación deseada que se quiere alcanzar y el proceso metódico es secuencia de instrucciones definida como plan.

En general, un plan consiste en una secuencia de uno o más pasos, cada uno de los cuales tiene un conjunto de atributos. Cada paso tiene el nombre del operador que debe ejecutarse y una obligación. La obligación es una lista de parámetros que el sistema debe pasar para alcanzar una determinada meta.

Un planificador está compuesto por estados, operadores, acciones. El estado son características que tiene un sistema en un instante de tiempo determinado. Los operadores son el medio, por el cual, el estado es modificado. Un operador toma como entrada un estado actual del problema, efectúa los cambios requeridos en el mismo, y devuelve el nuevo estado. Una acción es la instanciación de unos valores concretos en un operador.

Al igual que la mayoría de planificadores clásicos, necesitan una descripción del dominio y del problema, almacenada generalmente en ficheros de texto. El fichero del dominio muestra las primitivas necesarias, constantes, hechos y funciones para poder representar los estados del mundo, así como una descripción de las posibles acciones que pueden realizarse. El fichero del problema, por el contrario, define una instancia particular de problema, proporcionando una descripción del estado actual del Mundo y de los objetivos a conseguir.

Para describir estas situaciones se puede utilizar diferentes lenguajes de planificación. Uno de ellos es PDDL (Planning Domain Def nition Language) (21)

Los dominios PDDL contienen, esencialmente, la descripción del vocabulario del dominio (predicados, funciones, constantes) y los operadores de planif cación. Los operadores de planif cación, a su vez, se describen mediante precondiciones y efectos. Las precondiciones establecen las condiciones necesarias para poder aplicar la acción correspondiente, y los efectos describen los cambios que se producirán en el estado del sistema tras ejecutar la acción. Finalmente, los dominios PDDL contienen una descripción de las características que un planif cador debe tener para poder resolver problemas en este dominio. Usando este mecanismo, es fácil extender el lenguaje con nuevas características sin romper la compatibilidad hacia atrás.

Los problemas PDDL contienen una referencia al dominio que da contexto al problema, la descripción del estado inicial y los objetivos del problema. Los objetivos se suelen representar mediante las condiciones que un estado debe cumplir para ser considerado un estado objetivo, pero también es posible def nir objetivos más complejos imponiendo restricciones sobre la estructura de los planes, funciones de coste a minimizar, o incluso objetivos deseables pero no imprescindibles.

Capítulo 3. Análisis del Sistema

En este capítulo, se realiza una descripción del análisis que ha sido elaborado para la creación de un sistema de control, que realice mapas mediante el uso del robot P3DX y una Kinect. Se describirá el alcance del sistema, los casos de uso, los requisitos y las posibles alternativas para la realización del proyecto.

Este capítulo será de utilidad para sentar las bases de un diseño de sistema que proporcione la mejor solución.

3.1 Alcance del Sistema

El sistema que se quiere desarrollar, es un sistema capaz de generar mapas de entornos en los que las zonas navegables sean claramente definidas. Para ello, se crearán mapas que contengan la siguiente información:

- Un mapa que represente los obstáculos, las zonas desconocidas y libres del entorno.
- Un mapa de waypoints que contenga la información necesaria para que lo pueda usar un planificador. Esta información deberá tener datos como los espacios libres, la distancia que hay entre estos y un nombre que los identifique claramente. El fichero en el que se guarda esta información deberá ser de formato XML.

La construcción del mapa se realizará mediante el robot P3DX y la cámara Microsoft Kinect. El sistema tendrá que estar basado en el Framework ROS, para que otros sistemas con esta plataforma puedan adquirir los mapas generados. Otra cuestión por la que se usa ROS, es que todo el proceso de generación de mapas debe de ser visualizado y monitorizado por la herramienta Rviz de ROS. Véase Anexo ROS III.

Una vez que se hayan generado los mapas, el robot deberá navegar por ellos basándose en la información de estos.

3.2 Presentación del Servicio

En este apartado, se realiza una descripción breve del sistema que se quiere analizar en este capítulo. Esto ayudará a la hora de tomar los requisitos y analizar más las posibles soluciones al problema. Este sistema constará de tres subsistemas que son los siguientes:

- **Work-Station:** Este subsistema representa el sistema que ha sido desarrollado para este proyecto. En él, se encuentra implementado el sistema de control del robot y el sistema de generación de mapas, los cuales interactúan con los Cámara_Kinect y Pioneer 3DX. Estos obtendrá las referencias de los objetos del entorno y las medias del desplazamiento del robot. El subsistema Pioneer 3DX ejecutará las órdenes de desplazamiento y velocidad que Work-Station genere.

- **Cámara_kinect**: Es la encargada de captar la información de la posición de los distintos objetos de un entorno. Este subsistema enviará la información del entorno a Work-Station para que la procese.

- **Pionner3DX:** Este subsistema representa al robot P3DX, el encargado de realizar los movimientos basándose en las instrucciones del Work-Station. También mandará información de la posición basándose en su sistema odométrico.

En la *Ilustración 3.2-1* se muestra la interacción de estos bloques del sistema.

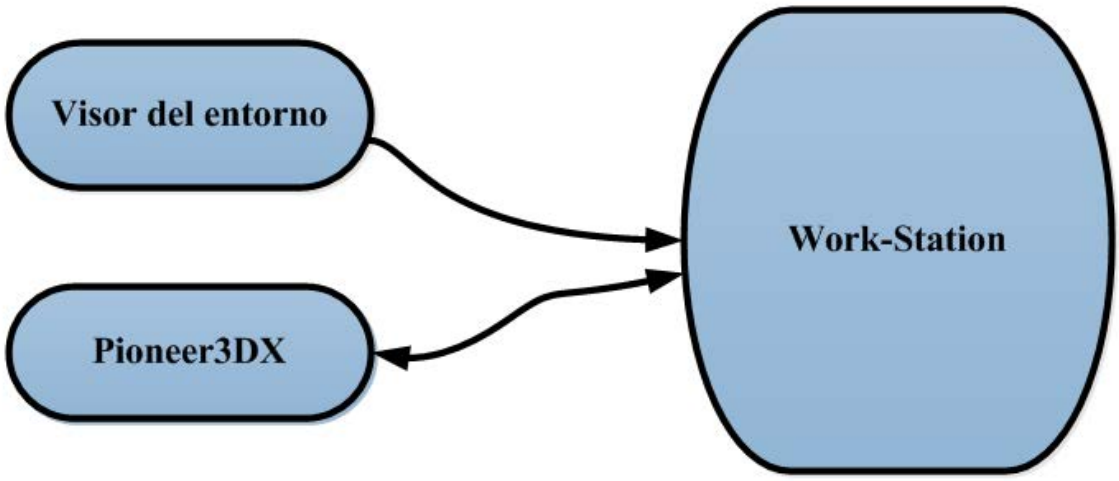

Ilustración 3.2-1. Modelo del Dominio.

Una vez realizada esta descripción del sistema, se pasará al estudio y análisis de las actividades que hará el sistema.

3.3 Modelo de Casos de Uso

En este apartado se describe los Casos de Uso a desarrollar. Los Casos de Uso son la descripción de las actividades que se deberán llevar acabo para la realización de un proceso. Las identidades que realizan los Casos de Uso se denominan Actores.

3.3.1 Descripción de los Actores

En este apartado del capítulo, se describen los actores que interactúan con el sistema, como se ha comentado. Los actores del sistema son aquellas identidades que realizan los Casos de Uso. Los Actores que interactúan con este sistema son los siguientes:

- **Robot**: Representa al robot P3DX, este actor interactúa con la aplicación, proporciona los datos que percibe del entorno y toma decisiones para seguir la trayectoria determinada por la aplicación. Aunque este actor se artificial, deberá tener la inteligencia para tomar decisiones en determinados momentos y mandar información a la aplicación, para que ésta pueda cumplir sus objetivos.

- **Usuario:** Representa al usuario encargado de modificar los datos de control de la aplicación.

Una vez identificado a los actores del sistema, hay que describir los Casos de Uso de la aplicación.

3.3.2 Descripción de los Atributos de los Casos de Uso

En este apartado, se realiza la descripción de cada uno de los atributos de los Casos de Uso. Es importante que queden bien definidos, para poder realizar una descripción de los Casos de Uso lo más completa posible. Los atributos de los Casos de Uso son los siguientes:

- Identificador: Identifica unívocamente el Caso de Uso**.**
- Nombre: Nombre del Caso de Uso.
- Descripción: Descripción de la funcionalidad del Caso de Uso.
- Actores: Actores relacionados con el Caso de Uso.
- Precondiciones: Acciones o hechos que se han de cumplir para que el Caso de Uso se pueda iniciar.
- Flujo normal: Serie de eventos que se desarrollan si el Caso de Uso se ejecuta sin problemas.
- Flujo Alternativo: Serie de eventos que se desarrollan, si se produce algún fallo al ejecutar el Caso de Uso. Puede que el Caso de Uso no tenga Flujo Alternativo.
- Pos condiciones: Acciones o hechos que se cumplirán si el flujo de eventos normal se ha ejecutado correctamente.

3.3.3 Diagrama de Casos de Uso

En este apartado, se define las relaciones entre los Actores y los Casos de Uso, como las dependencias que existen entre ellos y las asociaciones. Este diagrama está basado en la descripción de los objetivos del sistema de donde se ha sacado los Casos de Uso.

La *Ilustración 3.3-1* muestra el diagrama de Casos de Uso, describiendo las dependencias, asociaciones y extensiones de los Actores y Casos de Uso del sistema.

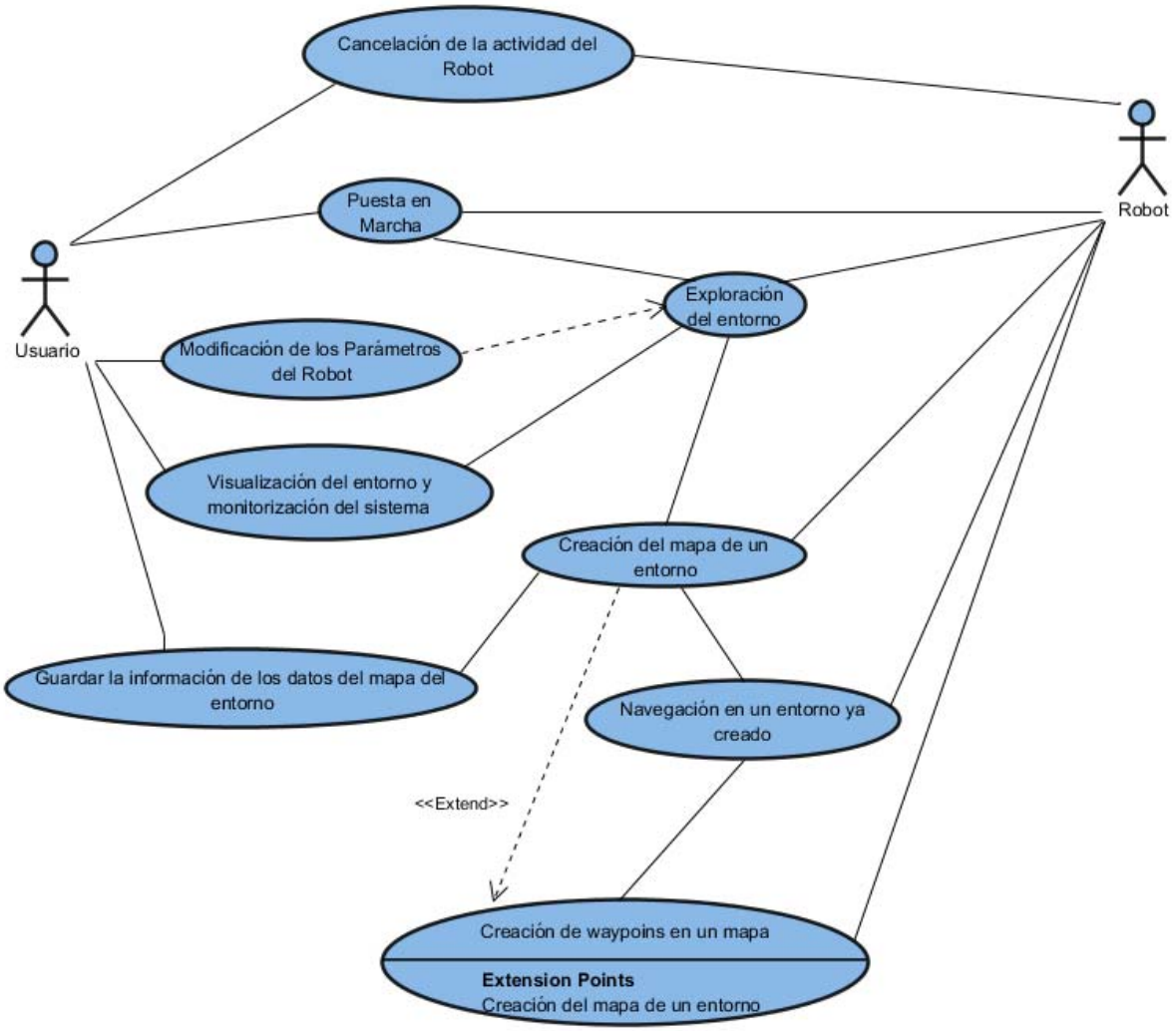

Ilustración 3.3-1. Diagrama de Casos de Uso.

Una vez analizados y descritos los Casos de Uso, solo queda sacar los requisitos del sistema. Por cada Caso de Uso saldrán uno o varios Requisitos del Sistema que deberá cumplir la aplicación a desarrollar.

3.3.4 Casos de Uso

Los casos de uso de esta aplicación son los siguientes:

Identificador	**CS01**	**Nombre**	**Puesta en marcha**
Descripción	Se establece las conexiones con todos los sensores del robot y se obtiene la información del sistema, que será actualizada durante la ejecución del programa.		
Actores	Robot, Usuario.		
Precondiciones	Los actores deben haber conectado todos los dispositivos del robot.		
Flujo Normal	1. El usuario abre una terminal del sistema operativo. 2. El usuario lanza la aplicación a través del comando Roslaunch. 3. El Robot verifica el estado de las conexiones con los sensores del robot.		
Flujo Alternativo	3. El sistema lanza un error a través de la terminal del sistema operativo.		
Pos condiciones	El sistema muestra información sobre el estado de las conexiones con los sensores del robot.		

Tabla 3.3-1. Caso de Uso 1.

Identificador	**CS02**	**Nombre**	**Visualización del entorno y monitorización del sistema.**
Descripción	Observación de la información que se percibe del entorno y del comportamiento del sistema.		
Actores	Usuario.		
Precondiciones	Los actores deben haber conectado todos los dispositivos del robot y haber comprobado el correcto funcionamiento de los mismos.		
Flujo Normal	1. El usuario abre una terminal del sistema operativo. 2. El usuario lanza la aplicación Rviz de ROS a través del comando `rosrun rviz rviz`. 3. Carga un fichero de configuración o aplica su propia configuración. 4. El observa el entorno a través de las ventanas de visualización de la aplicación Rviz.		
Flujo Alternativo	4. La aplicación Rviz no muestra nada a través de sus ventanas de visualización del entorno.		
Pos condiciones	El sistema muestra y guarda información sobre el entorno.		

Tabla 3.3-2. Caso de Uso 2.

Identificador	**CS03**	**Nombre**	**Exploración del Entorno.**
Descripción	El robot se desplaza por el entorno autónomamente con el objetivo de explorarlo.		
Actores	Robot.		
Precondiciones	Los actores deben haber conectado todos los dispositivos del robot y haber comprobado el correcto funcionamiento de los mismos.		
Flujo Normal	1. El usuario abre una terminal del sistema operativo. 2. El usuario ejecuta el nodo de navegación implementado de por el alumno con el comando `rosrun` o `roslaunch`. 3. El Robot mueve el motor y la trayectoria de la rueda		
Flujo Alternativo	1. El usuario puede navegar dando instrucciones al robot a través del teclado de la computadora.		
Pos condiciones	El sistema muestra y guarda información sobre el entorno. El robot se desplaza por el entorno.		

Tabla 3.3-3. Caso de Uso 3.

Identificador	**CS04**	**Nombre**	**Navegación en un mapa ya creado**
Descripción	Dado un mapa de un entorno, una posición inicial y un objetivo, el robot deberá ser capaz de navegar hasta el objetivo marcado en el mapa. El robot deberá ser capaz de reconocer la posición en la que se encuentra en el mapa.		
Actores	Robot.		
Precondiciones	Los actores deben haber conectado todos los dispositivos del robot y haber comprobado el correcto funcionamiento de los mismos. Necesitan a ver lanzado la aplicación ROS Rviz. Se necesita el mapa en el que se encuentra el robot y su posición inicial.		
Flujo Normal	1. El usuario abre un terminal. 2. Lanza el comando Roslaunch que lanza el proceso de navegación por el mapa. 3. Indica la posición en la que se encuentra el robot en el mapa. 4. Marca el objetivo.		
Pos condiciones	Los comportamientos del robot variaran dependiendo de los parámetros modificados.		

Tabla 3.3-4. Caso de Uso 4.

Identificador	**CS05**	**Nombre**	**Modificación de los parámetros.**
Descripción	Se modificará los parámetros del robot para cambiar el comportamiento de éste.		
Actores	Usuario.		
Precondiciones	Los actores deben haber conectado todos los dispositivos del robot y haber comprobado el correcto funcionamiento de los mismos. Necesitan a ver lanzado la aplicación ROS Rviz.		
Flujo Normal	5. El usuario abre un editor de textos 6. Edita los parámetros del archivo roslaunch. 7. Se compila el paquete donde se ha modificado el archivo roslaunch. Para compilar se utiliza el comando `rosmake`.		
Pos condiciones	Los comportamientos del robot variaran dependiendo de los parámetros modificados.		

Tabla 3.3-5. Caso de Uso 5.

Identificador	**CS06**	**Nombre**	**Creación del Mapa.**
Descripción	Se creará un mapa del entono donde se podrá observar las zonas libre de obstáculos y las que tiene obstáculos.		
Actores	Robot.		
Precondiciones	Se deberá ejecutar el proceso de navegación por el sistema		
Flujo Normal	1. Acceder a la terminal del sistema. 2. Lanzar el fichero .launch que contenga la información para la creación de un mapa, Este proceso se ejecutará con el comando `roslaunch`.		
Flujo Alternativo	1. Lanzar el proceso de creación de un mapa a través del comando `rosrun`.		
Pos condiciones	Se creará un mapa en formato png que podrá ser observado con la herramienta de ROS Rviz.		

Tabla 3.3-6. Caso de Uso 6.

Identificador	**CS07**	**Nombre**	**Creación de Waypoints.**
Descripción	Se creará un mapa del entono y sobre ese mapa se añadirá una serie de Waypoints para poder planificar rutas posteriormente.		
Actores	Robot.		
Precondiciones	Se deberá ejecutar el proceso de navegación por el sistema		
Flujo Normal	1. Acceder a la terminal del sistema. 2. Lanzar el fichero .launch que contenga la información para la creación de un mapa, Este proceso se ejecutará con el comando `roslaunch`.		
Flujo Alternativo	Acceder a la terminal del sistema y escribir el comando rosrun que lanza el ejecutable Waypoints.		
Pos condiciones	Se creará un mapa que podrá ser observado con la herramienta de ROS Rviz. Se creará un fichero XML que tendrá la información de la posición de los Waypoints. También tendrá la información de la posición inicial.		

Tabla 3.3-7. Caso de Uso 7.

Identificador	**CS08**	**Nombre**	**Guardar la información.**
Descripción	Se guarda la información que recrea el entorno y la información que se quiere añadir a ese entorno (los waypoints).		
Actores	Usuario.		
Precondiciones	Los actores deben haber conectado todos los dispositivos del robot y haber comprobado el correcto funcionamiento de los mismos. Necesitan a ver lanzado la aplicación ROS Rviz.		
Flujo Normal	1. El usuario abre una terminal del sistema operativo 2. Ejecuta el comando `roslaunch navegation.launch`.		
Pos condiciones	Se creará un fichero con extensión .yamp y png donde se guarda la información del entorno.		

Tabla 3.3-8. Caso de Uso 8.

Identificador	**CS09**	**Nombre**	**Cancelación de la actividad.**
Descripción	Se cancelará la actividad del robot y de la cámara Kinect.		
Actores	Robot, Usuario.		
Precondiciones	Se deberá estar ejecutando el proceso de creación de mapas o el de navegación por éstos.		
Flujo Normal	1. Acceder al terminal. 2. Pulsar los botones del teclado control-C.		
Flujo Alternativo	1. Cancelación de la actividad si surge algún imprevisto, en el que el sistema no pueda dar una solución 2. Impresión de mensaje de error en el terminal.		
Pos condiciones	Los comportamientos del robot variarán dependiendo de los parámetros modificados.		

Tabla 3.3-9. Caso de Uso 9.

3.4 Requisitos del Sistema

En este apartado del capítulo, se expondrán los requisitos software que debe cumplir el sistema para así poder llevar acabo todos los objetivos propuestos. Se realizará un catálogo de requisitos basado en tablas, que contendrán información de cada uno de ellos.

3.1.2 Definición de los Atributos de los Requisitos del Sistema

Los atributos de los requisitos del sistema son los siguientes:

- **Identificador:** Este contendrá la información por la que se ha catalogado el requisito, donde "RF" significa Requisito Funcional o "RNF" Requisito No Funcional, y "00" es el número de requisito en el catálogo. La funcionalidad de este apartado es la de reconocer unívocamente el requisito.

- **Caso de Uso:** Este campo identifica al Caso de Uso al que pertenece el requisito de software.

- **Descripción:** Contendrá información básica descriptiva sobre el requisito que se está definiendo.

- **Necesidad**: Nivel de necesidad del requisito. Su valor puede ser "Esencial", "Deseable" u "Opcional".

- **Prioridad**: Nivel de prioridad del requisito. Su valor puede ser "Alta", "Media" o "Baja".

- **Estabilidad:** Nivel de estabilidad del requisito, es decir, grado de variación que puede sufrir el mismo a lo largo del desarrollo del proyecto. Su valor puede ser "Alta", "Media" o "Baja".

- **Verificabilidad**: Sirve para medir la complejidad de su comprobación. Su valor puede ser "Alta", "Media" o "Baja".

- **Claridad**: Capacidad de descripción de la funcionalidad del requisito. Su valor puede ser "Alta", "Media" o "Baja".

- **Fuente**: Individuo que proporciona la restricción.

En los siguientes apartados están definidos en catálogo los Requisitos de Software Funcionales del Sistema y los Requisitos de Software No Funcionales del Sistema.

3.5 Definición de Requisitos Funcionales del Sistema

En este apartado se presenta el catálogo de los Requisitos Funcionales del Sistema, que son los requisitos que definen el comportamiento interno del software, como la forma de realizar cálculos, la forma de manipulación de los datos y otras funcionalidades que describen de manera práctica como llevar a cabo la implementación de los Casos de Uso. Las siguientes tablas representa el catálogo de Requisitos de Funcionales del Sistema:

Identificador	**RF00**	**Caso de Uso**		**CS06**	
Descripción	Creación de un servicio capaz de detectar imagen y posición de las marcas topológicas de referencia más importantes en el entorno de un robot autónomo móvil P3DX.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-1. RSF00.

Identificador	**RF01**	**Caso de Uso**		**CS06**	
Descripción	Obtener una imagen estructurada de los elementos del entorno.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-2. RSF01.

Identificador	**RF02**	**Caso de Uso**		**CS07**	
Descripción	Generar un mapa donde se sitúen los waypoints en lugares navegables para cualquier sistema robótico.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-3. RSF02.

Identificador	**RF03**	**Caso de Uso**		**CS01**	
Descripción	Crear un servicio que sea fácilmente utilizable por otros servicios implementados y sirva para lanzar la ejecución de todos los procesos.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-4. RSF03.

Identificador	**RF04**	**Caso de Uso**		**CS01**	
Descripción	Que el servicio sea fácilmente adaptable a nuevos entornos de trabajo en los que pueda operar el robot.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-5. RSF04.

Identificador	**RF05**	**Caso de Uso**		**CS05**	
Descripción	Los parámetros de navegación y exploración, así como el lanzamiento de dichos procesos y la definición de tamaño a explorar velocidades del robot deberán ser definidos por el usuario en un fichero que permita ser utilizado por el servicio principal.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-6. RSF05.

Identificador	**RF06**	**Caso de Uso**		**CS04**	
Descripción	Creación de un servicio de navegación por un mapa ya creado.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-7. RSF06.

Identificador	**RF07**	**Caso de Uso**		**CS08**	
Descripción	Los datos de salida del servicio consistirán en los mapas que contengan marcas topológicas de referencia detectadas y los Waypoints.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-8. RSF07.

Identificador	**RF08**	**Caso de Uso**		**CS04**	
Descripción	El servicio de navegación tiene que ser capaz de trabajar con imágenes ya procesadas del entorno en el que se realiza dicha navegación.				
Estabilidad	No estable	**Necesidad**	No esencial	**Prioridad**	Baja
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-9. RSF08.

Identificador	**RF09**	**Caso de Uso**		**CS01**	
Descripción	El servicio de navegación deberá ejecutarse de manera independiente de la creación de mapas y waypoints				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-10. RSF09.

Identificador	**RF10**	**Caso de Uso**		**CS01**	
Descripción	La aplicación deberá ser capaz de determina la posición del robot en el mapa en todo momento.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-11. RSF10.

Identificador	**RF11**	**Caso de Uso**		**CS03**	
Descripción	El servicio incluirá un sistema de exploración que permita al robot acercarse a una determinada distancia de una marca topológica.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-12. RSF11.

Identificador	**RF12**	**Caso de Uso**		**CS09**	
Descripción	Suspender todos los procesos al pulsar control-C				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-13. RSF12.

Identificador	**RF13**	**Caso de Uso**		**CS08**	
Descripción	Creación de un fichero que recoja la información del entorno de navegación y los waypoints marcados en ese mapa.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-14. RSF13.

Identificador	**RF14**	**Caso de Uso**		**CS01**	
Descripción	Lanzar todos los procesos con una sola instrucción en la línea de comandos, tanto para la Navegación como la creación de mapas.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-15. RSF14.

Identificador	**RF15**	**Caso de Uso**		**CS02**	
Descripción	El proceso de creación de mapas y la navegación por este debe ser visible en la herramienta de ROS Rviz.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-16. RSF15.

Identificador	**RF16**	**Caso de Uso**		**CS02**	
Descripción	El objetivo meta en la navegación será marcado a través de la herramienta Rviz de ROS.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-17. RSF16.

Identificador	**RF17**	**Caso de Uso**		**CS07**	
Descripción	El proceso de creación de waypoints deberá publicar información como la posición y el tiempo en que fue emitida.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-18. RSF17.

Identificador	**RF18**	**Caso de Uso**		**CS05**	
Descripción	Se podrá pasar por parámetro la superficie a procesar para la creación de Waypoints.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-19. RSF18.

Identificador	**RF19**	**Caso de Uso**		**CS05**	
Descripción	Se podrá a pasar por parámetro la disposición de los waypoints en el mapa.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-20. RSF19.

3.1.3 Definición Requisitos No Funcionales del Sistema

En este apartado, se presenta el catálogo de Requisitos No Funcionales. Estos requisitos se definen como aquellos que emergen de las funcionalidades del sistema como puede ser la fiabilidad, la capacidad, el procesamiento, la capacidad de almacenaje de datos o la respuesta en tiempo, al mismo tiempo, define las restricciones del sistema como la presentación de los datos.

Identificador	**RNF00**	**Caso de Uso**		**CS03**	
Descripción	El sistema no podrá realizar mapeados de entornos con pendientes.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-21. RSNF00.

Identificador	**RNF01**	**Caso de Uso**		**CS03**	
Descripción	La exploración será obligatoria realizarla con el sonar del Robot P3DX.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-22. RSNFS01.

Identificador	**RNF02**	**Caso de Uso**		**CS08**	
Descripción	El fichero que contiene la información de los waypoints tiene que estar en formato XML.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-23. RSNF02.

Identificador	**RNF03**	**Caso de Uso**		**CS08**	
Descripción	El fichero que contiene la información de los waypoints tiene que mostrar la información de la posición de los waypoints en un eje de coordenadas (X, Y), así como la distancia que hay entre los waypoints adyacentes.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-24. RSNF03.

Identificador	**RNF04**	**Caso de Uso**		**CS08**	
Descripción	El fichero de los waypoints tiene que tener la información necesaria para crear un dominio en PDDL.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-25. RSNF04.

Identificador	**RNF05**	**Caso de Uso**		**CS08**	
Descripción	El fichero creado para visualizar el entorno debe ser de formato. png.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-26. RSNF05.

Identificador	**RNF06**	**Caso de Uso**		**CS05**	
Descripción	Será obligatorio modificar los parámetros de navegación y creación del mapa en un fichero .launch.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-27. RSNF06.

Identificador	**RNF07**	**Caso de Uso**		**CS02**	
Descripción	El usuario no podrá realizar cambios en el código de la aplicación.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-28. RSNF07.

Identificador	**RNF08**	**Caso de Uso**		**CS03**	
Descripción	El robot dado un mapa y una posición inicial deberá ser capaz de navegar hacia otra posición marcada por el usuario en otro punto del mapa.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-29. RSNF08.

Identificador	**RNF09**	**Caso de Uso**		**CS01**	
Descripción	El código deberá estar optimizado para minimizar los tiempos de ejecución.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-30. RSNF09.

Identificador	**RNF10**	**Caso de Uso**		**CS01**	
Descripción	El servicio hará uso de bibliotecas externas para la parte de procesamiento de imagen.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-31. RSNF10.

Identificador	**RNF11**	**Caso de Uso**		**CS04**	
Descripción	El sistema de exploración del robot se basará en los datos del sonar para la detención de obstáculos.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-32. RSNF11.

Identificador	**RNF12**	**Caso de Uso**		**CS01**	
Descripción	Los datos de entrada del servicio serán proporcionados por la cámara Kinect de Microsoft para el modelado del entorno y detención objetos durante la navegación.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-33. RSNF12.

Identificador	**RNF13**	**Caso de Uso**		**CS01**	
Descripción	La aplicación tiene que ser capaz de funcionar en el entorno donde se realizan las pruebas.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-34. RSNF13.

Identificador	**RNFS14**	**Caso de Uso**		**CS03 C CS04**	
Descripción	El robot debe evitar la colisión con cualquier objeto del entorno.				
Estabilidad	Estable	**Necesidad**	Esencial	**Prioridad**	Alta
Verificabilidad	Alta	**Claridad**	Alta	**Fuente**	Tutor

Tabla 3.5-35. RSF20

3.6 Modelo del Dominio

En este apartado del capítulo, se va a realizar la descripción de un Modelo del Dominio. Un Modelo del Dominio, es un artefacto de la disciplina de Análisis, presentado como uno o más diagramas que contiene conceptos de la realidad física del Sistema. El Modelo del Dominio es un mapa conceptual utilizado por el analista para comprender mejor el sistema que se va a desarrollar.

El sistema a desarrollar en este proceso, se corresponde con el subsistema Work-Station presentado en la *Ilustración 3.6-1*. Éste se divide en cuatro subsistemas, los cuales, se describe de forma detallada a continuación:

- **Creación de Mapas:** Este subsistema es el encargado recrear la información del entorno, para ello deberá alcanzar todas las partes de éste y realizar una clasificación de los elementos que lo componen. Para la recreación del entorno, se basará en la información recogida del controlador, que le informará de la posición del robot en el entorno y del sistema Visor del entorno, que le mostrará la distribución de los objetos del entorno.

- **Navegación:** Este subsistema es el encargado de que el robot navegue por el mapa creado en el sistema de Creación de Mapas. Para ello, necesita de la posición del robot en ese mapa, que se la proporcionará el Visor del entorno y el Controlador, y la meta alcanzar, que será proporcionada por el usuario.

- **Controlador:** Es el encargado de los movimientos del robot y de dar la información del movimiento a los subsistemas de Navegación y Creación de Mapas. Este sistema interactuará con los sensores del robot para que se desplace sin colisionar con ningún objeto.

- **Visor del Entorno**: Es subsistema es el encargado de captar la información recogida del bloque Cámara_Kinect y enviársela a Navegación y Creación de Mapas.

En la *Ilustración 3.6-2* se muestra el flujo de intercambio de información explicado anteriormente.

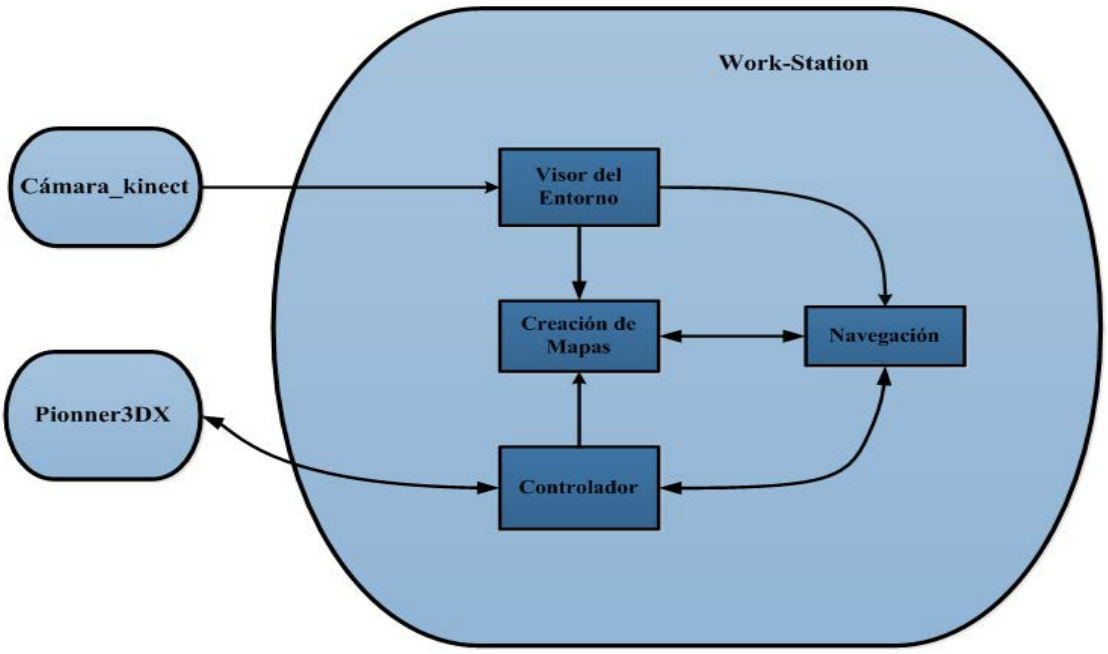

Ilustración 3.6-1. Subsistemas, Modelo del Dominio.

Este análisis del sistema a desarrollar, facilitará el análisis a la hora de valorar las diferentes alternativas del que se tiene para implementar el proyecto y lograr la solución óptima.

3.2 Análisis de las posibles soluciones

A continuación, se presentan las distintas alternativas posibles que existen para desarrollar el sistema descrito. Estas alternativas han sido definidas teniendo en cuenta el modelo de dominio y los requisitos descritos en este capítulo.

Teniendo en cuenta la información de productos los software del mercado, la descripción general del sistema, el estudio de la situación actual y el catálogo de requisitos, se va a redactar las diferentes alternativas que hay para configurar la solución, para su posterior estudio.

La descripción de las alternativas se basará en la descripción del Modelo del Dominio. Una vez analizadas las alternativas, se pasará a la valoración de riesgos de cada una de ellas. La valoración de riesgos se realizará basándose en:

- **Dependencias de subsistemas:** Debido a las fuertes dependencias que existen entre los distintos subsistemas, la aparición de un fallo en uno de ellos puede desencadenar en fallos en el resto. Es muy difícil prever cuando va a fallar un subsistema y por lo tanto, no se pueden ofrecer medidas para minimizar este riesgo.

- **Tiempo de Implementación:** El tiempo es un factor fundamental a valorar. El proyecto debe poder realizarse en un tiempo máximo definido por la complejidad del sistema a desarrollar.

- **Costes:** Las diferentes alternativas pueden suponer un coste adicional al proyecto. Es importante valorar este criterio, ya que es necesario reducir al máximo el precio total del proyecto. Este criterio va ligado al tiempo de implementación, ya que a mayor tiempo mayor gasto en recursos materiales y humanos.

- **Valoración de Riesgos:** Para una correcta valoración de las alternativas, es necesario tener en cuenta los riesgos que suponen cada una de ellas. Cuantos menos riesgos suponga una alternativa, más probabilidad tendrá de ser la solución óptima.

En el Anexo III. ROS, Anexo IV. Hardware del Sistema, se encuentra la descripción de los framework utilizados para el análisis de estas alternativas.

3.2.2 Alternativa 1

- **Controlador:** Esta alternativa consta de la implementación del controlador basándose en la librería "p2os" de ROS, esta librería proporciona los drivers[4] para conectarse con los robots de *"Adept MovilRobots" "MovilRobots"* y *"ActivaMedia"* (22), cabe

[4] Drivers. Es un programa que controla un dispositivo. Un driver actúa como traductor entre el dispositivo y los programas que utilizan el dispositivo.

destacar que también tiene un controlador para el robot basado su código de instrucciones en las letras del teclado de un ordenador.

- **Visor del Entorno**: En este caso el Sensor de Control de Operación es la cámara Microsoft Kinect, ya que está definido como un requisito. Un driver para esta cámara sería el proporcionado por *"OpenKinect"*. Esta comunidad proporciona código libre para realizar programas basados en la cámara Microsoft Kinect.

- **Navegación:** En este caso ROS proporciona una librería llamada *"PR2",* esta librería tiene el código para la navegación del P3DX. También consta de Move_base, que es una librería que planifica las trayectorias para alcanzar un estado meta.

- **Creación de Mapas**: Para este subsistema se puede emplear la creación de un código basado en los drivers de la librería "*ARIA*" esta librería, lo único que el código para esta librería está basada en el láser *Hockuyo* que es un complemento de *"ActivaMedia"* para el robot P3DX

Valoración de Riesgos de la Alternativa 1.

Esta alternativa supone los siguientes riesgos:

1) La realización del controlador basado en "*p2os_drivers*", puede suponer un coste de tiempo muy elevado, ya que habría que implementar todo el movimiento del robot basándose en los datos que proporcionan estos drivers.

2) La utilización de los drivers proporcionados por la comunidad *"OpenKinect"* supone también un coste grande en tiempo, ya que hay que ajustarlo a ROS.

3) El controlador del robot se basa en la herramienta Rviz de ROS, si se realiza con el código de *"ActivaMedia",* habría que reajustarlo a ROS, por lo cual, supondría un coste elevado de dinero y tiempo de implementación.

3.2.3 Alternativa 2

- **Controlador:** Para la implementación de este subsistema se puede utilizar la librería *"Aria",* que contiene una gran cantidad de funcionalidades para los robots de "*MobileRobots/ActivaMedia*" (22) al cual, pertenece el P3DX. Una de estas funcionalidades es el control de velocidad, giro y detención de obstáculos.

- **Navegación:** Para la navegación se puede utilizar el stacks de ROS *"turlebot_navigation",* este stacks proporciona el código Move_base y el cambio de la nueve de puntos en 3D a 2D, cosa que es muy útil, ya que uno de los requisitos es el mapa en 2D. El stack que proporciona este cambio se llama *"costmap_2D"*.

- **Visor del Entorno:** El driver para la interfaz con la cámara Kinect será *"Openni_Kinect",* que es un stack proporcionado por ROS y este se puede ajustarse a cualquier sistema basado en ROS.

- **Creación de Mapas:** Se puede utilizar la librería *"Gmapping"* que es otro stack de ROS, esta librería proporciona la implementación del algoritmo de Filtrado de Partículas. Es fácil y adaptable a cualquier sistema basado en ROS.

Valoración de Riesgos de la Alternativa 2.

Esta alternativa supone los siguientes riesgos:

1) El tiempo de coste para la implementación del controlador del robot con *"Aria"* sería medio debido a que muchas funcionalidades están implementadas, el único problema que surge es que hay que reajustarlas a ROS, aunque esto no supone un gran problema, ya que tanto ROS como ARIA (22), son bastante portables y están implementados en el mismo lenguaje que es C++.

2) En la navegación surge un problema y es que la implementación para este subsistema es para el robot "*TortleBox*" y no para los robots "*MobileRobots/ActivMedia,* y puede suponer un riesgo a la hora de implementar dependencias entre subsistemas.

3.3 Solución Elegida

Se puede deducir, de la valoración de alternativas realizada, que la solución óptima se corresponde a la **Alternativa 2.** Esta alternativa es la que menos riesgos supone, además sus riesgos son asumibles, tanto en coste de tiempo y dinero. Aunque en esta solución se podrá aplicar una solución parcial de la Alternativa 1. La solución parcial que se pueden aplicar es la siguiente:

- En la creación del subsistema de navegación el stack Move_base proporciona un perfecto funcionamiento y se adapta a Rviz perfectamente, el único problema que surge que esta librería manda en las instrucciones de movimiento y giro basadas en ROS. Por lo que si el controlador se basa en "*Aria*" supone un gran problema de ajuste. La solución parcial que se puede aplicar es realizar el subsistema en "*Aria*" y la navegación en "*p2os_drivers*" que está basado en ROS y es perfectamente ajustable a "*Move_base*".

En el siguiente capítulo, se realizará una descripción más detallada de la solución a implementar. Explicando esta solución desde un punto de vista mas técnico.

Capítulo 4. Diseño de la Solución Técnica

En este capítulo del documento se explicará la solución que se ha diseñado. Se realizará la presentación de la arquitectura del sistema. En esta presentación se describe los componentes del sistema desarrollado, así como la manera en la que interactúan entre ellos, en los diferentes procesos.

Se describirá la forma en la que se debe desarrollar las funcionalidades del sistema, como la creación de waypoints y la creación de marcos de referencia.

Este capítulo permitirá realizar una implementación más concisa y ordenada del sistema. El diseño de la solución del sistema se ha basado en el capítulo de Análisis. Esto proporcionará que la solución diseñada contenga todas las especificaciones fijadas en el análisis.

4.1 Arquitectura del Sistema

En esta parte se realizará una descripción del sistema que se ha implementado. Definiéndose cada una de las partes y sus correspondientes funcionalidades, así como la manera de interactuar unas con otras. Cabe destacar, que esta arquitectura esta compuestas por componentes[5], para realizar una descripción más estructurada.

Basándose en el capítulo de análisis, en el que se realizó una primera aproximación a la estructura del sistema, se realizará la descripción de cada una de las partes del sistema, de menos a más detalle de abstracción. La *Ilustración 4.1-1* muestra la arquitectura del sistema dividida en subsistemas, componentes e interfaces del sistema, así como las asociaciones que establecen unos con otros.

[5] Componente. Es un conjunto de patrones y abstracciones coherentes que proporcionan el marco del programa.

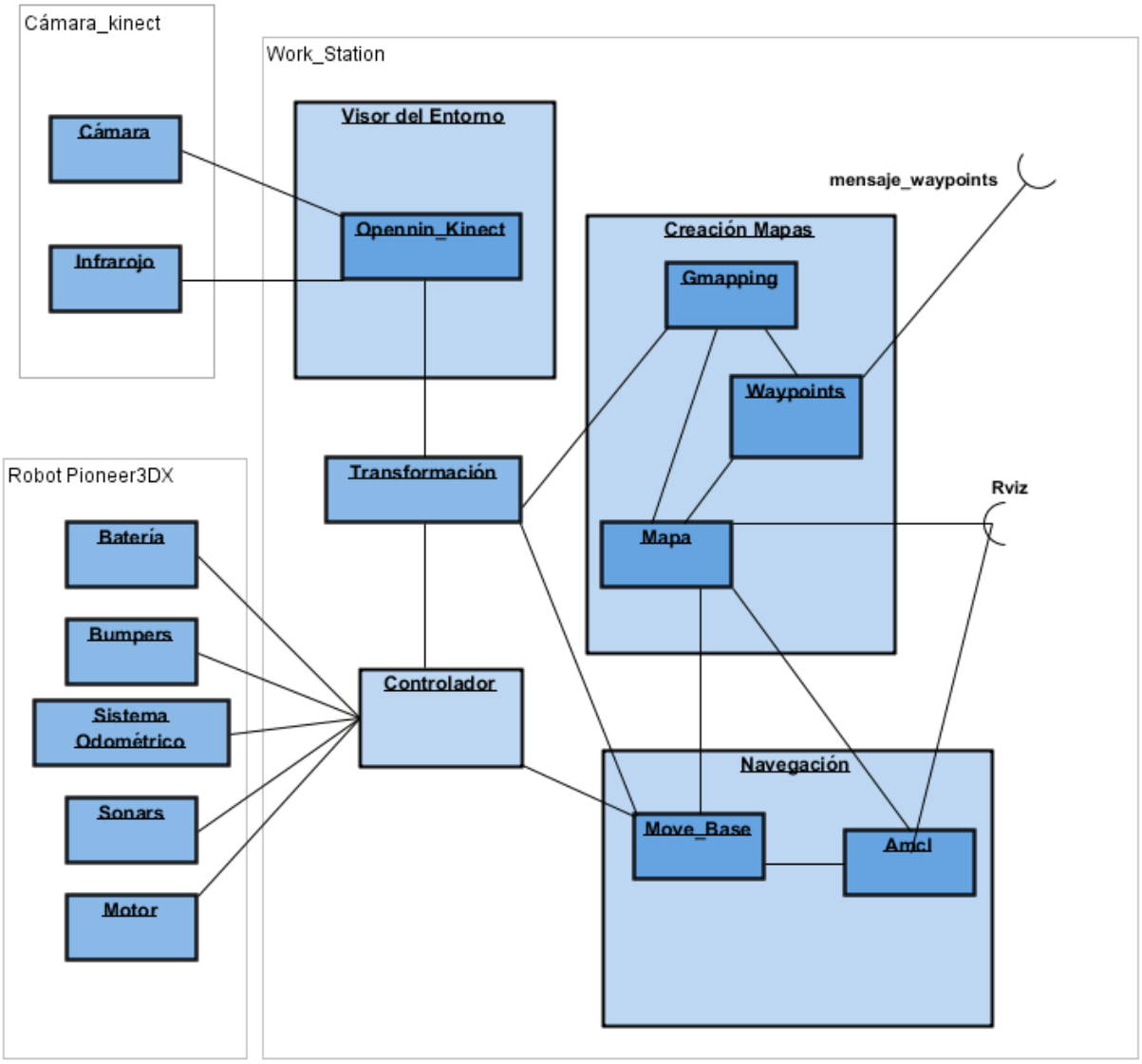

Ilustración 4.1-1. Arquitectura del Sistema.

La *Ilustración 4.1-1* muestra gráficamente la arquitectura del sistema. Los elementos del diagrama son los siguientes:

- **Robot Pionner3DX:** Es el encargado de procesar la información de los sensores para mandársela al bloque de Work-Station. También se encarga de conectar y desconectar dichos sensores, pero siempre bajo la supervisión del subsistema Work-Station. Estos componentes se conecta al controlador a través del puerto ttyUSB0. También tiene como funcionalidad, procesar los datos para modificar las trayectorias del robot. Los componentes de este sistema son los siguientes:

  - **Batería:** Este componente se encarga de mandar información de la energía que se dispone**.**

- **Motor:** Es el componente encargado del desplazamiento del robot, tendrá dos estados encendido o apagado. Cuando este encendió podrá moverse por el entorno.

- **Sonar**: Recibe información de la posición de los objetos que sirve para modificar la trayectoria del robot y así poder evitar los obstáculos.

- **Sistema Odométrico:** Este componente es el encargado de procesar los vectores de dirección y posicionamiento del robot.

- **Bumpers**: Este componente es un sensor de contacto, que en caso de que el sonar no detecte un obstáculo, este será el encargarlo de detectarlo y mandar la información al bloque Work-Station, para que este pueda procesar una respuesta.

- **Cámara Kinect:** Este subsistema es el encargado de obtener las distancias a los obstáculos del entorno. Con los datos recibidos por este bloque, se intentará recrear el entorno en el que se encuentra el robot. Estos componentes se conectan al Visor del entorno a través del puerto USB.

  - **Infrarrojo:** Se encarga de medir las distancias con los objetos. La diferencia que tiene este componente, con el Sonar es la de recrear el entorno. Esta información se la pasará al subsistema Visor del Entorno, para que pueda transformarla en información útil.

  - **Cámara:** Su función es la de obtener la información de los componentes del entorno, como los objetos y las formas, aunque estas últimas pueden modelarse también con el Láser.

1. **Work-Station**: Este subsistema se encarga de procesar la información de los subsistemas Robot Pionner3DX y Cámara Kinect. Es el principal subsistema y en el que se basará el desarrollo. Si se quisiera realizar una colaboración con este sistema debe ser a través de Workstation. Work-Station es la base todo el sistema, porque es aquí donde se realizan todas las funcionalidades. Este subsistema está compuesto por las siguientes componentes

   - **Controlador:** Este componente se encarga de procesar la información del robot, recibe la información a través de un puerto USB, la procesa y da una respuesta. La información que comparte con los demás componentes, son las coordenadas de desplazamiento del robot en el mapa. Además de estas funcionalidades, será el encargado de interactuar con los componentes del subsistema Robot Pionner3DX. Al componente Motor le pasará la señal de activación y desactivación. Del Sonars obtendrá la información de los objetos, en forma de array de 8 posiciones, cada una de ellas representa a los sonars que se muestran en el Anexo IV Hardware del Sistema. De los bumper recibirá una señal de activación en caso de colisión con algún obstáculo. El Sistema Odométrico proporcionará las distancias y direcciones que va tomado el robot, estas distancias serán utilizadas para el desarrollo de los mapas y de la navegación por ellos. De la batería obtendrá la cantidad de energía disponible, esto servirá de ayuda para guardar los datos en caso de que se esté agotando.

Este componente se encargará de recibir las órdenes de desplazamiento del subsistema de navegación y las aplicará al robot.

- **Visor del entorno:** Este componente tiene la funcionalidad de establecer la conexión con el subsistema Cámara_Kinect, a través de un puerto USB. Tiene como objetivo obtener toda la información que se capte de la cámara y del infrarrojo de la Kinect. Deberá realizar una estructuración de esta información para que pueda ser útil a los diferentes componentes que la necesiten.

  - **Openni_Kinect:** Tiene la funcionalidad de procesar la información de la distancia con los objetos para mandársela a Gmapping. Esta información deberá ser procesada por el componente Transformación para adecuarse a la situación de la Kinect con el robot. Este componerte se encargará de estructura la información para que se forme un mapa en dos dimensiones. Otras de sus funciones, es obtener la gama de colores de los objetos del entorno, así como las diferentes distancias a las que están. Para ello, se ayudará de la información de los componentes Infrarrojo y Cámara del subsistema Cámara _Kinect.

- **Transformación**: Este componente se encarga de realizar la transformación y rotación de los marcos de referencia del robot con el de la cámara Kinect, ya que el sistema odométrico y la posición de la Kinect no están inscritos bajo el mismo marco de referencia. Se hará una descripción más detallada de este componente en el apartado Creación Marcos de referencia de este capítulo.

- **Creación_mapas:** Este componente es el encargado de crear los mapas del entorno y generar los waypoints con la información del componente Transformación y Opennin_Kinect.

  - **Waypoint:** Este componente es el encargado de marcar los waypoints en las zonas libres de obstáculos del mapa. Para la creación de estos waypoints, se utilizará una rejilla, que dividirá el espacio a gusto del usuario. Para obtener una mayor descripción de este componente, es necesario realizar una lectura del apartado Creación de Waypoints de este capítulo.

  - **Gmapping:** Este componente del sistema se encarga de procesar la información recibida de la cámara Kinect y las coordenadas de la posición del robot tomadas del servicio Controlador**.** Con ello, genera los datos necesarios para hacer una representación del entorno. Este servicio se basa en el algoritmo del filtro de partículas para representar el entorno. Antes de obtener las posiciones de los componente Controlador y Opennin_kinect, será necesario que se aplique las diferentes cambios de referencia en los marcos de referencia de cada uno d ellos, así se podrá realizar una representación del entorno más correcta. Como se ha comentado anteriormente, de esto se encargará el componente Transformación.

  - **Mapa:** Es el componente encargado de crear la imagen del mapa, este mapa marcará las zonas libres en blanco, las desconocidas en gris y las ocupadas en

negro. Además de ello, deberá guardar la posición inicial de la que partió el robot y un valor de la resolución en metros/pixel en la que ha creado el mapa.

- **Navegación:** Este componente se encarga de la navegación de robot por un entorno basándose en la información que crea el componente Creación_mapas y la posición del robot.

  - **Move_base:** Es el encargado de ejecutar el plan de navegación. Este componente se encarga de crear un plan para alcanzar una meta. Pasa los parámetros de movimiento al componente Controlador, para guiarle a la meta. La distribución de los objetos en el entorno, serán obtenidas de Openni_Kinect. A la representación de estos objetos les hace un ensanchamiento para que el robot no colisione y entenga un margen de error en caso de mala medición de los sensores. En la *Ilustración 4.1-2* el polígono irregular de lados rojos que se observa, pertenece a la representación de la posición del robot, los pixeles rojos a los obstáculos, y los azules al ensanchamiento de los obstáculos.

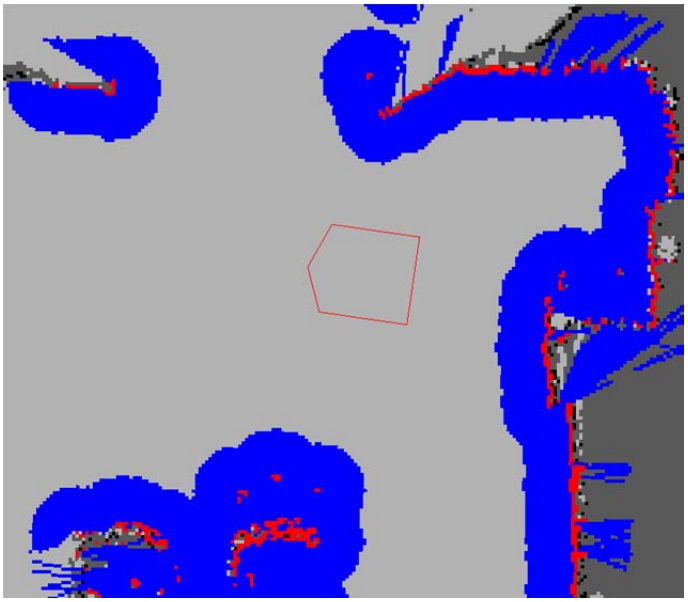

Ilustración 4.1-2. Inflado de Obstáculo, Move_base

Se basará de los componentes Mapa y Waypoint, para hacer una representación del entorno en el que se quiere realizar la navegación

  - **Amcl:** Es el encargado de situar las posiciones de la meta y del robot en un ínstate de tiempo. Para ello, se basará en la implementación del método de Monte Carlo, explicado en el punto 2.6.1.4 del capítulo Estado de la Cuestión.

- **Mensaje_waypoints:** Este componente se encarga de crear una interfaz por la que cualquier sistema basado en ROS, pueda obtener la información de los waypoints de un mapa. Esta información será los waypoints del mapa, las conexiones de éstos, el tiempo en el que se envió y si el mapa ha sido terminado.

- **Rviz:** Es una interfaz gráfica, por la cual se podrá visualizar y monitorizar los proceso de creación de mapas y navegación, que se explicarán en el siguiente apartado.

4.2 Creación de Marcos de referencia para el sistema

En este apartado se describe los marcos de referencia que se comentaban en el capítulo Estado de la Cuestión, apartado 2.5. Estos marcos de referencia deberán implementarse en este sistema así como las conexiones entre ellos. Estos marcos son parte de las funcionalidades que realiza el componente transformación. Véase el a Anexo III. ROS

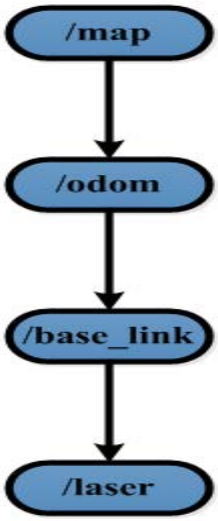

Ilustración 4.2-1. Árbol de Marcos de Referencia del Proyecto.

Como se puede apreciar en la *Ilustración 4.2-1*, en este sistema existen cuatro marcos de referencia que son los siguientes:

- **/map:** Representa al marco de referencia fijo del mapa. Este marco de referencia servirá para representar las coordenadas de los Waypoints.

- **/odom:** Representa la posición del robot según el sistema de odometría. Los vectores de referencia de este marco son corregidos por los algoritmos de localización del paquete Gmapping proporcionado por ROS. La conexión entre el marco de referencia /map y /odom, se hace a través de la clases Amcl y Slam_gmapping, que se describirá en el Anexo V Diseño detallado de la Aplicación.

- **/base_link:** Este marco de referencia pertenece al centro de la base móvil del robot, como se puede apreciar en la *Ilustración* realiza una conexión con el marco de referencia /odom, aunque esto son los mismos en el caso de P3DX. La conexión entre estos se hace a través de la clase tf_congf proporcionada por ROS, esta clase lo que hace es lo explicado en el apartado 5.4.

- **/laser:** Corresponde al marco de referencia donde se encuentra montado la Kinect. La relación entre /base_link y /laser se encuentra descrita en la clase Pionner_tf que está descrita en el siguiente apartado. Una vez, que se realiza la transformación entre el marco de referencia /base_link y /laser, si se quiere obtener las posiciones del láser será necesario llamar a /base_link.

La creación de marcos de referencia es imprescindible, si no la creación de mapas y la navegación por estos serán erróneas, ya que no hay una concordancia entre las posiciones de estos.

4.3 Interacción entre Componentes del Sistema

En este apartado, se describe la forma en la que interactúan los distintos componentes de la arquitectura descrita anteriormente y la información que se mandan unos a otros.

Como la Creación de Mapas y la navegación son procesos distintos, en la que los componentes interactúan de manera diferente, se dividirá este apartado en dos subapartados, que serán Creación de Mapas y Navegación.

Para la Creación de Mapas, el robot no necesitará ejecutar un plan, se moverá por el entorno esquivando los objetos, hasta que reciba por parte del usuario la orden de parar este proceso o hasta que se termine de construir el mapa. Además, en este proceso el sistema tendrá que ir construyendo el mapa y los waypoints, ya que el entorno es totalmente desconocido para él. Mientras que en la navegación el entrono si se conoce y lo que hace falta es ejecutar un plan de ruta.

4.3.1 Creación de Mapas

El este proceso será el que cree los mapas de los entornos. Para ello, representará la información en dos mapas distintos, que son los siguientes:

- Mapa Normal: En él se plasma la información de la distribución de los objetos del mapa, así como las zonas libres de ellos. Si no se conoce una zona del entorno, se establecerá como desconocida. Este mapa será creado por el componente Mapa.

- Mapa Waypoints: Este mapa muestra la disposición de los waypoints en las zonas libres de obstáculos y las conexiones entre estos.

La construcción de este mapa se realizará a través de la información recogida del sistema odométrico y el infrarrojo de la cámara Kinect. Esta información será tratada por los componentes controlador y Openni_Kinect respectivamente. Una vez que ha sido tratada por estos controladores, se enviará al componente Transformación para que establezca las referencias de las posiciones. Con estas posiciones Gmapping realizará una estimación de la disposición de los objetos.

La *Ilustración 4.3-1* muestra el diagrama de secuencia de los componentes del sistema en el proceso de Creación de Mapas.

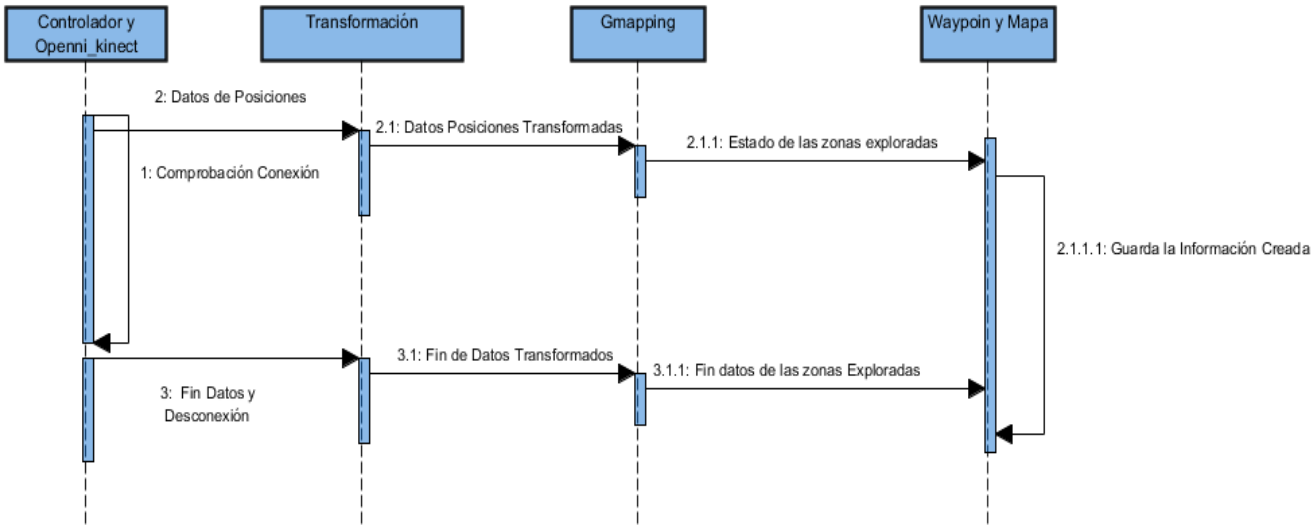

Ilustración 4.3-1. Iteración de Componentes, Creación de Mapas.

En la *Ilustración 4.2-1* se puede observar que sean unificado componentes, esto se ha realizado así para simplificar el diagrama de secuencia. Los componentes que interviene en el proceso de Creación de Mapas son los siguientes:

1) **Controlador y Openni_Kinect**: Estos dos componentes son los primeros en intervenir, establecen las conexiones con los componentes del Robot-P3DX y la Cámara Kinect respectivamente. Una vez que han establecido la conexión, manda los datos recogidos por estos componentes a Transformación. Estos datos contiene la distancia con los objetos del entorno y la posición en la que se encuentra el robot. Una vez que se ha terminado o ha habido una desconexión con alguno de los componentes del sistema, éste termina de mandar datos y muestra un mensaje de desconexión.

2) **Transformación**: Este componente transforma los datos de las posiciones, ya que la Cámara Kinect está colocada físicamente en otra posición que la del sistema de odometría del robot. Una vez transformado los datos, se mandan al componente Gmapping. Este último, realiza las transformaciones hasta que se dejan de mandar los datos del Controlador y de Openni_Kinect.

3) **Gmapping**: Recibe los datos transformados y aplica el algoritmo de filtrado de partículas descrito en el capítulo Estado de la Cuestión. Este componente manda la información de las zonas libres, ocupadas y desconocidas a los componentes Mapa y Waypoints.

4) **Waypoint y Mapas**: Estos componentes reciben los datos de Gmapping y crean la imagen y establecen los waypoints. Cuando reciben un mensaje de fin de datos, guardan la información creada y ponen fin al proceso de generar información. Cabe destacar, que los datos se guardan cada cierto tiempo, hasta el mensaje fin de datos explorados, mandado por el componente Gmapping. Estos crearán los mapas y waypoints hasta que el Controlador mande un mensaje de Fin.

4.3.2 Navegación

En este apartado se realiza una descripción de la interacción de los componentes del Sistema a la hora de realizar el proceso de navegación. El proceso de navegación se produce cuando una vez realizada la creación de mapas de un entorno, al robot se le pasa una meta y éste tiene que alcanzarla. La meta será un waypoint de los mapas de waypoints, construido en el proceso de creación de mapas. Basándose en la meta y la disposición de los objetos descrita en el Mapa Normal, el robot deberá alcanzar esa meta sin colisionar con ningún obstáculo. La manera de alcanzar esa meta, tiene que ser la que menos distancia requiera para ello. Como se observa no se podrá realizar este proceso hasta que no se haya generado un mapa del entorno.

Un punto en común que tiene la navegación con la Creación de Mapas, es que los dos necesitan transformar las posiciones de referencia, para que lo que lean los sensores concuerde con la realidad.

En este proceso los componentes que intervienen son a diferencia de la creación de mapas Navegación y Move_base. La intervención de estos componentes hace que la interacción sea diferente que en la creación del mapa, ya que ahora se necesita ejecutar un plan creado por el componente Move_base.

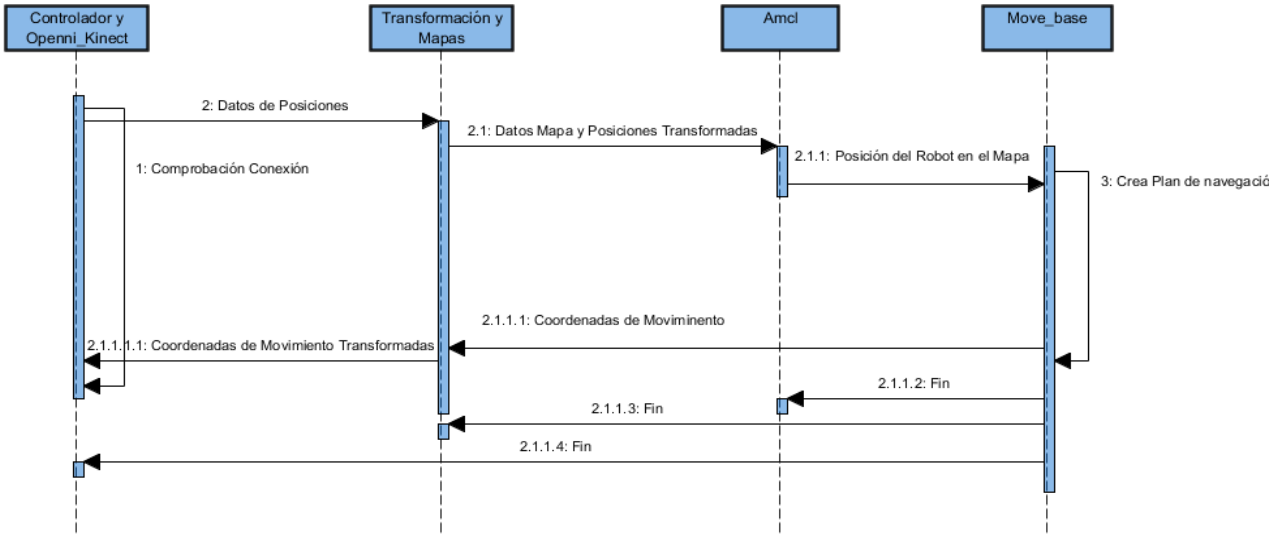

Ilustración 4.3-2. Iteración de Componentes, Navegación.

La interacción entre los componentes que se muestra en la *Ilustración 4.3-2* es la siguiente:

1) **Controlador y Openni_Kinect**: Estos componentes mandan las posiciones del robot y los objetos al servicio de transformación. También realizan una comprobación cada cierto tiempo de la conexión de los componentes.

2) **Transformación:** Realiza la transformación de los datos recibidos de los componentes Controlador y Openni_Kinect y se lo manda al componentes de Navegación.

3) **Amcl**: Con los datos transformados se establece la posición actual del robot en el mapa. Una vez establecida la posición del robot en el mapa, se manda esta información al componente Move_base.

4) **Move_base:** Este componente crea un plan para alcanzar el estado meta, en el que se debe situar el robot. Para alcanzar la meta, este componente manda las coordenadas de movimiento a Transformación, para que éste las transforme y se las mande al Controlador. Cuando se haya alcanzado la meta Move_base mandará un mensaje de fin a los componentes que intervienen en esta actividad.

4.4 Creación de Waypoints

En este apartado del capítulo se describe la forma en la que se asigna los waypoints en un mapa. Para ello, se describirá una rejilla que divida el mapa en zonas, cada una de estas zonas se les llamará celdas. Las celdas serán de un tamaño definido por el usuario. El tamaño se define por el alto y por el ancho de estas celdas en metros.

Cuando se implemente la rejilla habrá que basarse en los pixeles de la imagen que da Gmapping, este componente dará los pixeles que están ocupados, libres o desconocidos para él. Por lo tanto, para construir la rejilla habrá que basarse en los siguientes elementos:

- **Pixeles**: El componente Gmapping da una resolución de 0.05 m2/ pixeles. Por lo cual, el tamaño de un pixel será de:
    - lado pixel = $\sqrt{0.05}$ = 0.2236 m
- **Coordenadas**: Para asignar una posición al waypoint, se recorrerá la imagen desde la esquina inferior izquierda de la imagen, hasta la esquina superior derecha. Por cada pixel que se recorra se incrementará los metros de X e Y.
- **Tamaño de la celda**: Este será definido por el usuario, pero habrá que hacer un redondeo a lado pixel, ya que se recorre por pixel. Este redondeo se hace tomando la función suelo de la división del taño de la celda entre lado pixel.
- **Posición centro de la rejilla**: El centro de la rejilla será la división del lado de la celda entre dos. A continuación, realizará la función suelo de esta división y se sumará la mitad de lado pixel. Está operación se realiza para los dos lados de la celda.
- **Estados de los pixeles**: Según Gmapping tiene tres estados para los pixeles:
    - Ocupado: Que le asignará un valor de [0, 0.1).
    - Libre: Que tendrá un valor de (0.1, 0.65].
    - Desconocido: Que tomará un valor de [0.1, 0.65].
- **Estado de la celda**: El estado de la celda será ocupado, desconocido o libre, si todos los pixeles que integran la celda toman estos valores respectivamente.

La *Ilustración 4.4-1* representa la rejilla de un mapa donde los puntos simbolizan a los waypoints. La cuadrícula en negro con las coordenadas representa la posición en la que se encuentra el robot en el mapa. Esta posición la ofrece un servicio de ROS llamado *Pose.position*.

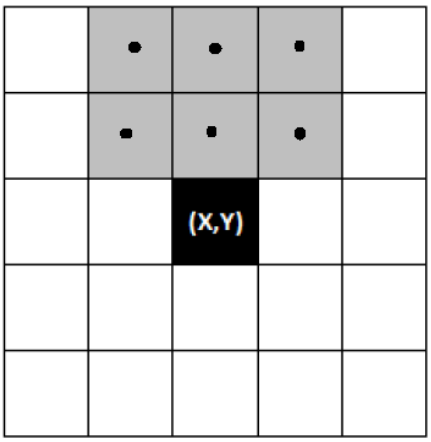

Ilustración 4.4-1. Rejilla.

La *Ilustración 4.4-2* muestra el proceso de creación de la rejilla y la fijación de los waypoints en el espacio libre de los entornos.

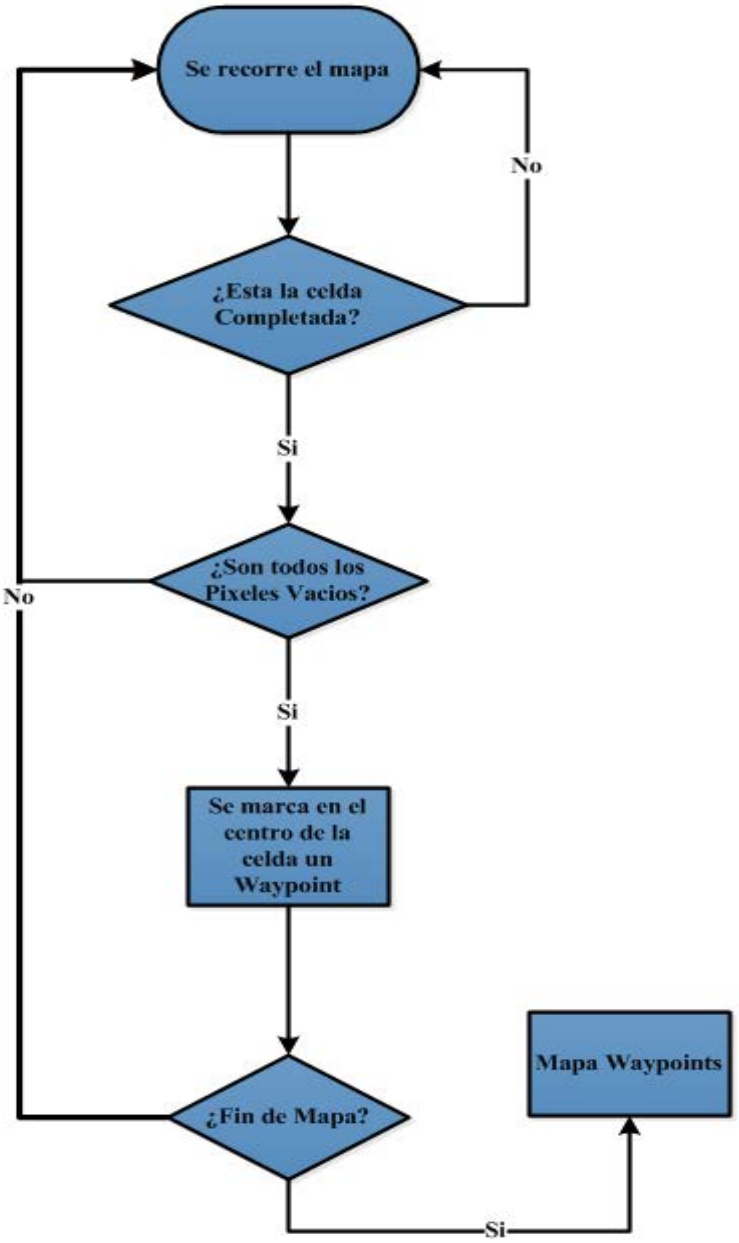

Ilustración 4.4-2. Ejecución del Algoritmo Waypoint.

El algoritmo que crea la rejilla realizará los siguientes pasos:

1. Se recorrerá la imagen pixel por pixel, desde la esquina inferior izquierda hasta la esquina superior derecha, contado los pixeles que están ocupado, libres o son desconocidos.
2. Se comprobará si se ha recorrido toda una celda de la rejilla.
3. Si se ha recorrido, se comprueba que todos sus pixeles estén en estado libre.
4. Si los pixeles están en estado libre, se asigna un waypoint.

5. Si se llega a la esquina superior derecha, fin al proceso de creación del mapa.

Este proceso creará los waypoints para un dominio PDDL, esta información queda recogida en un fichero XML. La información de este fichero es la siguiente:

- **Entorno**: Este será el entorno a representar. Tendrá los siguientes atributos:

  - Nombre del entorno: El nombre del entorno representado.
  - Unidades: Las unidades en las que se miden las distancias.
  - y: Alto del mapa.
  - x: Ancho del mapa.
  - Área total: Área total del mapa.
  - Área total cubierta: Área reconocida del mapa, se mide contado los pixeles ocupados y libres.

- **Robot**: Representa al robot, sus atributos son proporcionados por ROS que los obtiene del sistema de odometría del robot y la trasformación del marco de referencia del mapa, los atributos son los siguientes:

  - Nombre: El nombre del robot.
  - X: Posición del robot en el eje de las X
  - Y: Posición del robot en el eje de las Y.
  - Orientación: Posición hacia la que está dirigida la parte frontal del robot.

- **Waypoint**: Representa al waypoint en el mapa, los atributos para representar este elemento son la siguiente:

  - Nombre: En el fichero XML se representa como Code.
  - Posición_X: Posición en el eje X del waypoint.
  - Posicion_Y: Posición en el eje Y del waypoint.
  - Conexión: Este atributo muestra la conexión entre waypoint y la distancia que hay entre ellos.

La *Tabla 5.4-1* muestra el ejemplo de un fichero XML para un mapa de cuatro waypoints.

```
<?xml version="1.0" ?>
<Map>P3DX-Mapa
  <Datos_Mapa structure="Mapa-domain" name="Mapa-domain" units="metros" y="24.000000" x="14.400000" area="345.600010" area_covered="70.250001" />
  <robot name="P3DX" x="12.200000" y="1.000000" orientacion_x="0.000000" orientacion_y="0.000000" orientacion_w="1.000000" />
  <waypoin code="1" posicion_X="2.900000" posicion_Y="10.700000">
    <connection code="4" distance="0.650000" />
  </waypoin>
  <waypoin code="2" posicion_X="1.600000" posicion_Y="11.350000">
    <connection code="3" distance="0.650000" />
    <connection code="4" distance="1.300000" />
  </waypoin>
```

```
   <waypoin code="3" posicion_X="2.250000" posicion_Y="11.350000">
      <connection code="2" distance="0.650000" />
      <connection code="4" distance="0.650000" />
   </waypoin>
   <waypoin code="4" posicion_X="2.900000" posicion_Y="11.350000">
      <connection code="1" distance="0.650000" />
      <connection code="2" distance="1.300000" />
      <connection code="3" distance="0.650000" />
   </waypoin>
</Map>
```

Tabla 4.4-1. Fichero XML

La *Ilustración 4.4-3* muestra la imagen del mapa al cual corresponde el fichero XML representado en la *Tabla 4.4-1.*

Ilustración 4.4-3. Mapa del entorno

Capítulo 5. Pruebas Realizadas

Este capítulo, se describe las pruebas realizadas al sistema. Las pruebas tiene por objeto la verificación del cumplimiento de los requisitos, tal y como se describen en la especificación de requisitos software, las especificaciones de los Casos de Uso y la especificación de requisitos suplementarios.

5.1 Descripción de las Pruebas

El proceso de pruebas tiene por objeto alcanzar los siguientes objetivos concretos:

- Identificar la mayor cantidad posible de defectos del software lo antes posible.
- Identificar posibles defectos e inconsistencias en las especificaciones de este sistema.
- Verificar que el software entregado cumple con las especificaciones de requisitos definidos en el capítulo de Análisis.

Para que las pruebas puedan alcanzar la verificación de los requisitos, es necesario seguir unos procedimientos en las distintas fases de vida del proyecto. Estos procedimientos son los siguientes:

- **Inspección de Instalación:** Este procedimiento consiste en la inspección manual de la plataforma de pre-producción, tras la ejecución de las actividades de instalación. Este procedimiento queda plasmado en el Anexo I. Manual de Instalación.

- **Ejecución de las Pruebas:** Este procedimiento consiste en la ejecución de uno o varios casos de prueba, en la plataforma de pre-producción y la observación de los resultados.

En este capítulo del proyecto se describirá el último punto, para ello se identificarán las siguientes situaciones por prueba:

- **Creación del mapa del entorno**: En esta parte se comprobará que el sistema es capaz de identificar las marcas topologías del entorno y plasmarlas en un mapa de forma correcta.

- **Creación de Waypoints:** Los waypoint necesitarán tener total correlación con el mapa creado, identificando a las zonas libres y a los Waypoint adyacentes, para ello se utilizará una rejilla con las convenientes medidas.

- **Ejecución de un Plan de navegación:** El objetivo es probar que el robot es capaz de crear un plan de navegación basándose en el mapa creado del entorno. Para ello, se utilizará un observador ajeno al proyecto, al cual, se le realizará una serie de preguntas para comprobar esta funcionalidad. Este observador tendrá que ser como mínimo estudiante de alguna ingeniería industrial, informática o de telecomunicaciones, ya que a este grupo podrán pertenecer los posibles usuarios. La idea de aplicar un observador ajeno al proyecto, es darle otro punto de vista e intentar identificar más posibles errores. Este test recoge las siguientes cualidades del software que debe tener esta funcionalidad:

    - Eficiencia: El software tendrá que consumir los menos posible los recursos Hardware.
    - Robustez: El sistema deberá reaccionar bien a situaciones imprevistas.
    - Usabilidad: El software debe de ser fácil de usar y de entender para los usuarios que podrían interactuar con esta aplicación. Para ello se utilizará como observador a un graduado en ingeniería informática.

Se realizarán tres pruebas al sistema para comprobar que se cumple todo lo anterior. En estas pruebas queda recogido todo lo descrito anteriormente.

5.1.1 Descripción Prueba 1

La primera prueba se realiza en la galería orientación oeste del edificio Sabatini del Campus de Leganés. En esta prueba, se cubrirá situaciones en las que el robot navegará y explorará en entornos abiertos y sin obstáculos. Otro caso, sería ver cómo actúa el robot a la hora de moverse por este tipo de lugar tan abiertos y sin demasiada complejidad en la creación un mapa. Se pretende ver como realiza el proceso de Creación de Mapas descrito en el capítulo Diseño de la Solución, para ello se comprobará como genera los mapas siguientes:

- Mapa Normal: El problema que puede surgir a la hora de representar entornos abiertos, es que el infrarrojo de la cámara Kinect, al tener un alcance de tres metros, podrá dejar de recrear zonas en las que no llega.

- Mapa de Waypoints: Las celdas de la rejilla en las que se fijan los waypoints deberán ser casi todas las adyacentes a los waypoints marcados, intentando que no se dejen islotes de celdas entre celdas marcadas con un waypoint y quedando marcado el espacio libre en la mayor medida de lo posible, dado el tamaño de celda establecido por el usuario.

Para la realización de esta prueba se establecerá un ancho de celda grande de 1.5m x 1.5m, así se podrá observar cómo se establecen los waypoints cuando están alejados y como se ajustan las celdas al espacio libre cuando éstas son grandes.

Ilustración 5.1-1. Plano Galería.

La *Ilustración 5.5-1* anterior, muestra el plano de la galería donde se va a realizar esta prueba. El punto rojo muestra el lugar donde se empezó la exploración. La parte de la izquierda de la galería es la que da al patio del edificio de Sabatini, y la parte derecha a los despachos B14, B12, B10, B08, B06, B04 y laboratorio 2.1.B16, 2.1.B18.

5.1.1.1 Resultado Prueba 1

En el mapa creado por la clase Map_server durante el proceso de creación de mapas, como se observa, se puede distinguir las puertas de los despachos 2.1B.16, 2.1.B.18, el pasillo que tiene una puerta de dos hojas, y los tragaluces de los despachos B14, B12, B10, B08, B06, B04. Cabe destacar, que en el otro lado de la galería es una cristalera sin salida, por lo cual, se muestra una línea casi continua de pixeles en estado ocupado. Este mapa queda plasmado en la *Ilustración 5.1-2.*

Los laterales del pasillo delimitan el espacio abierto, como es en realidad. Sin embrago, el final del pasillo no aparece delimitado, ya que la exploración se paró antes de llegar, debido a que el cable de la corriente eléctrica se quedó sin recorrido. Durante el proceso de depuración de código de la clase Waypoints, los tragaluces aparecían como zonas abiertas, esto era debido a que entraba demasiada luz por ellos. En esta prueba aparece cerrados, ya que a la hora en la que se hizo, la luz solar tenía menos intensidad, haciendo que el infrarrojo de la Kinect recogiera mayor información sobre los objetos (22) (24).

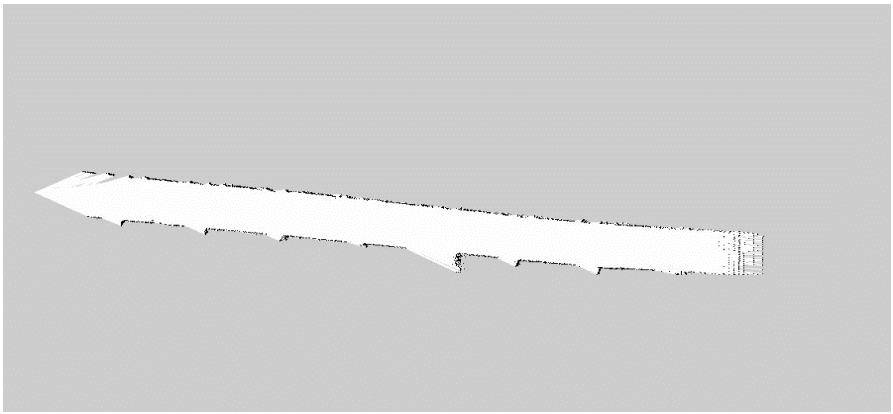

Ilustración 5.1-2. Mapa de la Galería

La *Ilustración 5.1-3* muestra la rejilla creada para realizar esta prueba, cuyas celdas son de 2.25 m^2. Las celdas cuyos pixeles sean todos blancos deberán tener un waypoint en el centro de estas, que será recogido en el fichero XML y en el mensaje que se emita la interfaz waypoint_mensaje. Se contarán los Waypoints plasmados en el XML y se compararán con las casillas en blanco de

la rejilla para comprobar que hay el mismo número de waypoints y las conexiones se realizan correctamente, siendo todas con los waypoint adyacentes y formado un grafo conexo. Para facilitar este proceso, se marcarán las casillas que están totalmente en blanco con un punto negro. Todo este proceso queda representado en la *Ilustración 5.1-3*

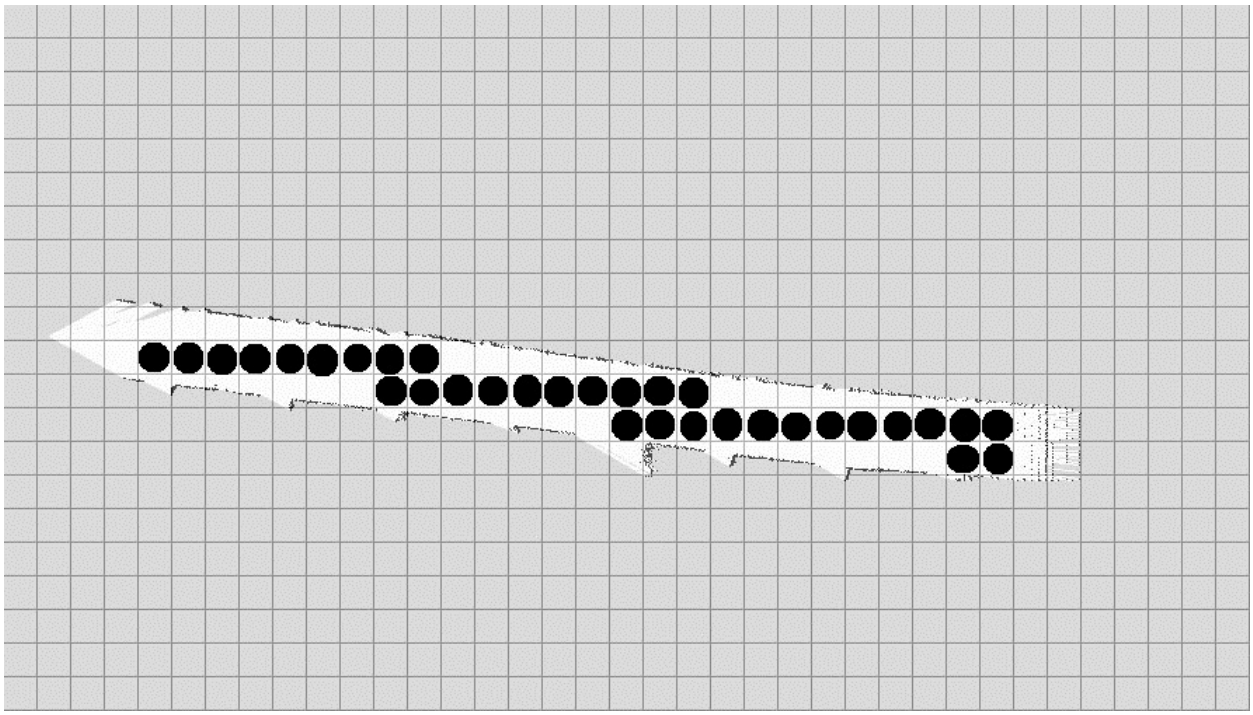

Ilustración 5.1-3. Rejilla de la Galería.

Para comprobar que el robot puede seguir un plan de navegación, el usuario deberá ejecutar el plan de actuación sobre el mapa de la Galería y contestar a las siguientes preguntas que se muestran en la *Tabla 5.1-1*. El plan de ejecución es el que se muestra en la *Ilustración 5.1-4*, siendo el estado de partida, el punto de la izquierda y el final el puto del a derecha.

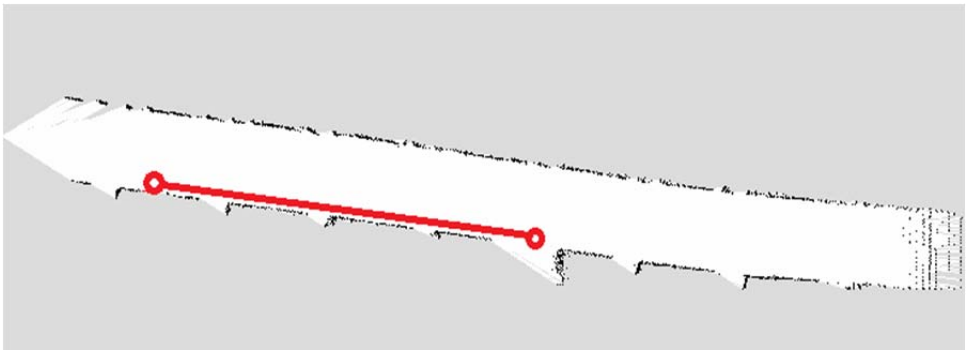

Ilustración 5.1-4. Plan de Ruta, Prueba1 de navegación.

Pregunta	Respuesta		Descripción de la incidencia
	Si	No	
¿El robot alcanza el objetivo marcado en el mapa?	x		
¿El programa Rviz muestra algún error?		x	
¿El programa lanza algún error por consola?		x	
¿El robot ha realizado la ruta óptima?	x		

Tabla 5.1-1. Test de Navegación. Prueba1.

5.1.1.2 Conclusiones de la Prueba 1

Como se observa en la creación del mapa en la *Ilustración 5.1-2*, la imagen corresponde con la de la galería plasmada en el esquema, además como se hizo en una hora del día en al que la luz del sol no era intensa, la zona de la cristalera que da al patio, se mostró como una zona cerrada. Cabe destacar que si se realiza a medio día, algunas zonas de la cristalera se mostrarán como abiertas.

En la *Ilustración 5.1-2,* se ve que la distribución de waypoints no es correcta, ya que queda demasiado espacio libre sin marcar. Para solucionar este problema, se debería reducir el tamaño de la celda. Lo que se aprecia, es que todos waypoints están conectados, haciendo que se forme un grafo conexo.

En la *Ilustración 5.1-3.* Muestra la ruta de navegación que realizará el robot. En la *Tabla 6.1-1*, se muestra las preguntas que el usuario ha contestado y no se puede apreciar ninguna incidencia reseñable.

5.1.2 Descripción Prueba 2

Esta prueba se realizará para ver cómo actúa el robot en lugares estrechos, en los que apenas cabe. Además, también se pretende comprobar si los datos que plasma en el mapa, son correctos y si comete cierto fallo a la hora de establecer los waypoints, ya que sería la parte más compleja debido al poco espacio libre.

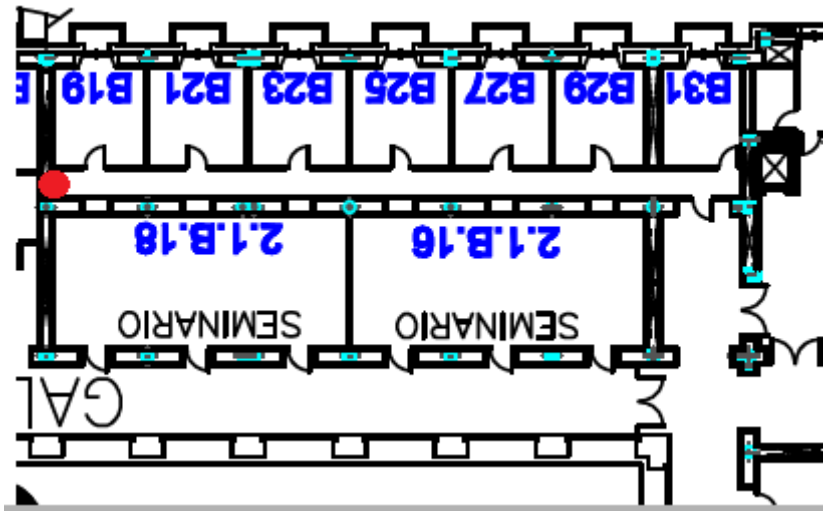

Ilustración 5.1-5.Plano Pasillo Estrecho

El objetivo de la prueba, es ver si el robot cuando explora, es capaz de llegar al final del pasillo y salir hacia la galería, en la que se encuentra las entradas a los laboratorios 2.1.B16 y 2.1.B18, como muestra la *Ilustración 5.1-5*. En la *Ilustración 5.1-5* se puede ver un punto rojo, este será el lugar de partida de la exploración.

5.1.2.1 Resultado Prueba 2

Como se aprecia en la *Ilustración 5.1-6*, el robot ha llegado hasta el final del pasillo. No muestran las puertas, ya que todas ellas estaban cerradas, pero si se puede apreciar que un lateral está mejor definido que otro. Esto se debe a que uno tiene una pared y en el otro están los cercos de las puertas.

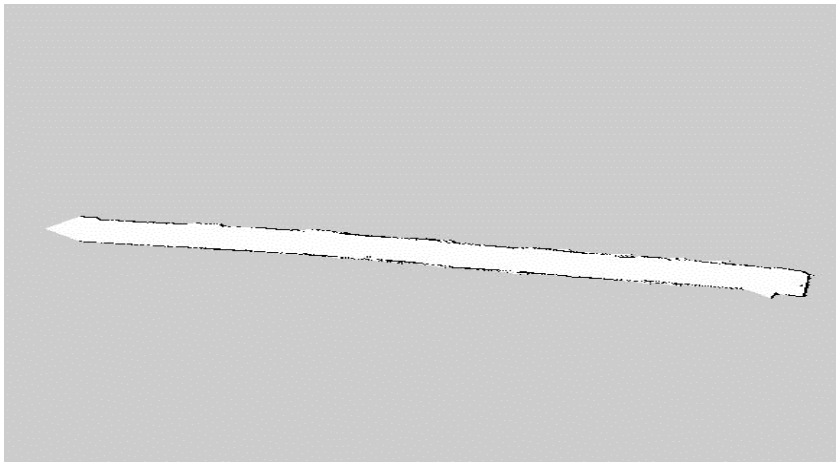

Ilustración 5.1-6. Mapa Pasillo Estrecho. Prueba2.

En la *Ilustración 5.1-7*. Se muestra la rejilla para el pasillo estrecho. Cabe destacar en este punto, que esta prueba se realizó con una rejilla más grande y no se marcó ningún waypoint, ya que ninguna celda contenía todos sus pixeles en estado libre. La celda era de 2.25 m^2, siendo su alto y ancho de 1.5 m. La *Ilustración 5.1-7* muestra una celda de 0.5 cm X 0.5 cm

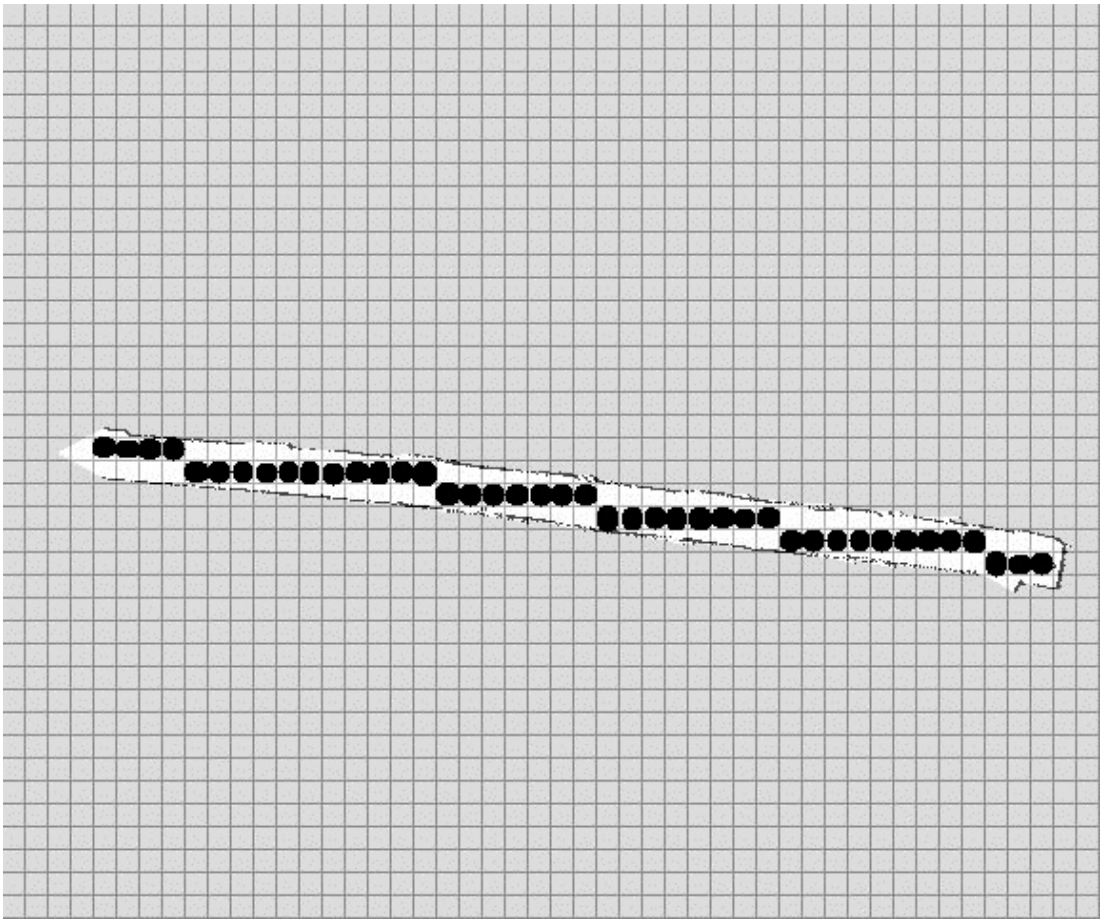

Ilustración 5.1-7. Rejilla Pasillo Estrecho. Prueba2.

Para comprobar que el robot puede seguir un plan de navegación, el usuario deberá ejecutar el proceso de navegación sobre el mapa del pasillo estrecho y contestar a las preguntas que se muestran de la *Tabla 6.1-2*. Si cree que es conveniente modificar los parámetros del láser para que el robot navegue mejor, deberá hacerlo. Se ha fijado como estado final la puerta que da a la galería, para tener un punto de referencia más exacto a la hora de comprobar los resultados de esta prueba.

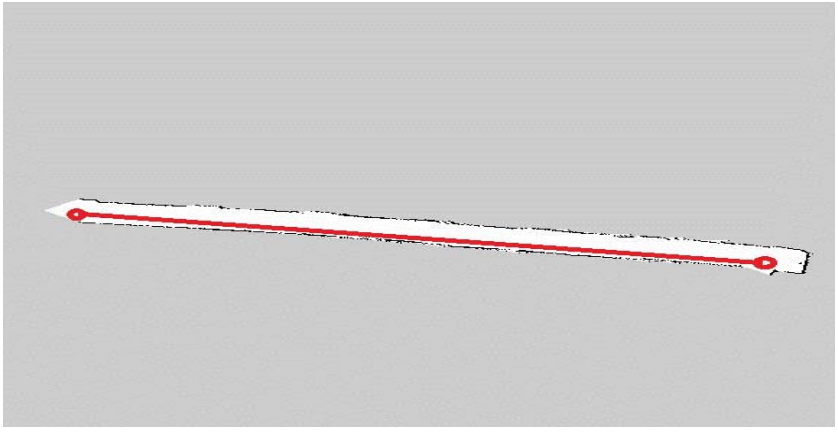

Ilustración 5.1-8. Plan de navegación. Prueba2

Pregunta	Respuesta		Descripción de la incidencia
	Si	No	
¿El robot alcanza el objetivo marcado en el mapa?	x		
¿El programa Rviz muestra algún error?		x	
¿El programa lanza algún error por consola?		x	
¿El robot ha realizado la ruta óptima?	x		

Tabla 5.1-2. Test de navegación. Prueba2.

5.1.2.2 Conclusiones de la Prueba 2

Como se puede apreciar en la *Ilustración 5.1-6*, no se ha llegado a la Galería, como se especificó en la descripción. Esto ha sido debido, a que el robot llego al final del pasillo y al encontrarse obstaculizado por las tres paredes, realizo un giro de 180º y regreso a la posición de inicio.

También, se puede apreciar que hay cierta desviación en el pasillo y éste no es totalmente recto. La causa de que suceda este hecho es por lo siguiente:

- Una junta de dilatación: En mitad del pasillo hay una junta de dilatación del edificio, que no está del todo a lineada con el suelo, haciendo que haya un pequeño badén de 3 cm de ancho, que hace que el robot haga un movimiento brusco y se mueva el láser de la cámara Kinect. Para resolver este problema, basta con fijar la Kinect a la estructura del P3DX.

Cabe destacar, que este entorno no será navegable para robots grandes por las características físicas que tiene, por lo cual, si se quiere realizar una marcación de waypoints habrá que realiza una rejilla como máximo de 0.2 m x 0.2 m

La navegación se realizado al segundo intento, ya que hubo que modificar la distancia de seguridad que estaba establecida para el láser. A pesar de esto, el robot llego a su estado meta realizando los menos movimientos posibles y describiendo su trayectoria con una línea recta, cosa que no sucedió en la exploración que se hizo de lado a lado del pasillo.

5.1.3 Descripción Prueba 3

Esta prueba se realizará en un pasillo con columnas en el centro. Lo que se pretende con esto es ver cómo actúa el robot en entornos con muchos obstáculos. La exploración se ha comenzado en

el punto rojo marcado en la *Ilustración 5.1-9*. Esto se ha realizado así, para evitar que el robot salga hacia la galería a través de la puerta de doble hoja situada en el lado izquierdo del despacho B14, ya que lo que se intenta en esta prueba es que plasme un entorno con obstáculos. En la *Ilustración 5.1-9,* se ve el plano del entorno donde se ejecutarán las pruebas.

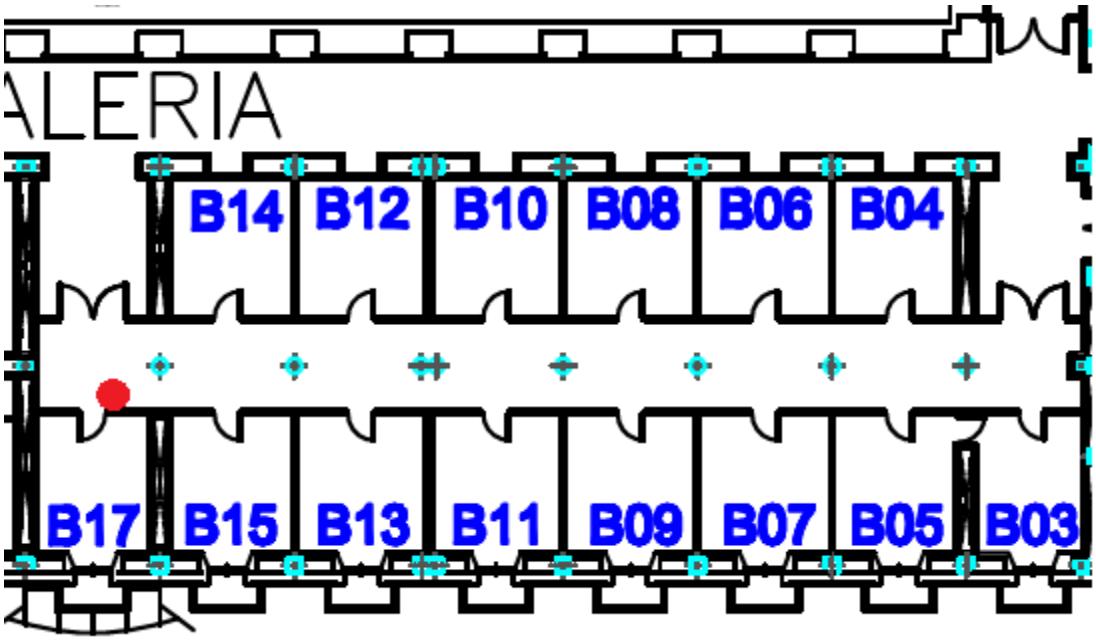

Ilustración 5.1-9. Plano de Pasillo con Columnas.

En esta prueba, el robot debe esquivar las columnas marcadas con una cruz en el plano de la *Ilustración 5.1-9.* El problema que puede surgir, es que al esquivar las columnas, quede espacio del mapa sin representar, debido a que el robot para esquivarlas las rodea por un lado nada más.

5.1.3.1 Resultado Prueba 3

Como se aprecia en la *Ilustración 5.1-10,* la primera columna no se muestra, esto se debe a lo comentado en la descripción y a que el haz de la nube de puntos no alcanza a la columna. Otra cosa que se aprecia en el mapa, es que las columnas no están bien definidas, esto es debido a que el robot salió por la puerta de doble hoja situada al lado derecho del despacho B04, por lo que debería a ver realizado dos pasadas, para completar mejor el mapa y no lo hizo. El robot debería a ver vuelto al punto de partida para obtener una mejor definición de las columnas.

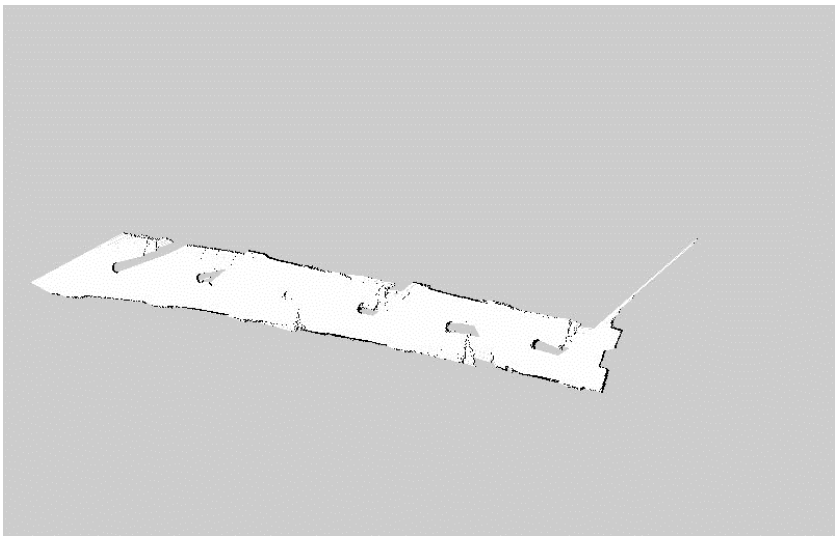

Ilustración 5.1-10. Mapa Pasillo de Columnas. Prueba3.

En esta parte se ha fijado una rejilla de tamaño igual al de las pruebas anteriores con el objetivo de ver si se realiza correctamente, esta rejilla es la mostrada en la *Ilustración 5.1-11*, Aquí sí que hay espacio suficiente para el tamaño de celda 0.5m X 0.5m, el cual, hará que se cree un grafo conexo de waypoints, representando en mayor medida de lo posible las zonas libres del entorno.

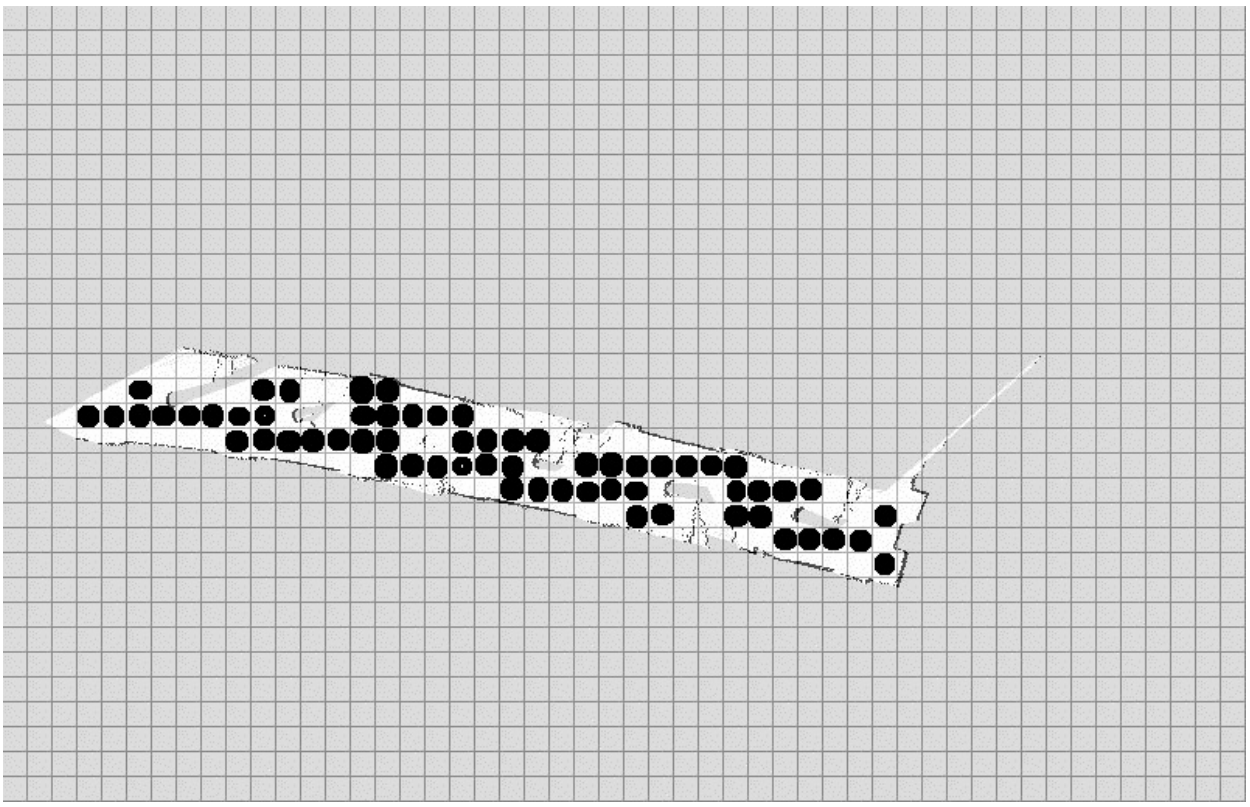

Ilustración 5.1-11. Rejilla Pasillo de Columnas. Prueb3.

Para comprobar que el robot puede seguir un plan de navegación, el usuario deberá ejecutar el plan de actuación sobre el mapa del pasillo con columnas, como muestra la *Ilustración 5.1-12* y

contestar a las siguientes preguntas que se muestran en la *Tabla 5.1-3*. Se deberá prestar mucha atención a los movimientos del robot y ver que éste se mueve de manera eficiente para alcanzar el estado meta, que será la puerta de doble hoja que está al lado del despacho B14, representada en la *Ilustración 5.1-9*.

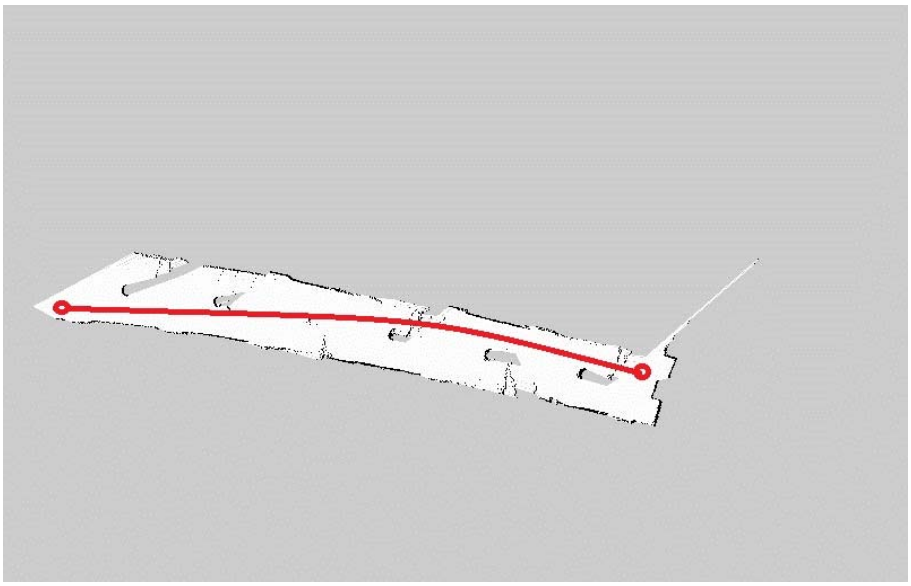

Ilustración 5.1-12. Plan de navegación. Prueba3.

Pregunta	Respuesta		Descripción de la incidencia
	Si	No	
¿El robot alcanza el objetivo marcado en el mapa?	x		
¿El programa Rviz muestra algún error?		x	
¿El programa lanza algún error por consola?		x	
¿El robot ha realizado la ruta óptima?	x		

Tabla 5.1-3. Test de navegación. Prueba3.

5.1.3.2 Conclusiones Prueba 3

En la creación del mapa, las columnas no se han definido correctamente, ya que el infrarrojo ha realizado una exploración solo por las partes delanteras y laterales, por eso no quedan bien definidas. El robo debería haber realizado una pasada durante el proceso de exploración por ambas partes, pero esto ha sido imposible, ya que salió hacia la galería, a travesando la puerta que se encuentra al lado derecho del despacho B14.

En una primera ejecución de la prueba, se pudo apreciar que los waypoint no estaban unidos todos con todos, no eran un grafo conexo. Esta situación no se puede tomar como errónea ya que hay que mirarlo desde otro punto de vista y sacar la siguiente conclusión:

- El pasillo de columnas será navegable parcialmente para robots grandes. Estos no podrán llegar a todos los punto de este entorno.

También se puede sacar otras dos conclusiones, éstas tienen que ver con modificar los parámetros de la rejilla:

- Crear una rejilla con celdas más pequeñas, haciendo imposible la navegación de robots grandes.

- Creación de otro tipo de rejilla, que no tendría celdas en forma polígonos regulares sino irregulares que se adapten a los obstáculos del entorno y la ubicación de los waypoint pasaría a estar lo más alejada posible de los obstáculos. Cabe destacar, que esto no garantiza la movilidad de robots grandes, debido a que podrían surgir colisiones laterales con los obstáculos.

En la *Ilustración 5.1-11* se muestra una rejilla con celdas de 0.25 m X 0.25 m, con lo que queda solventado el problema comentado anteriormente.

La navegación se ha realizado correctamente, realizando la menor cantidad posible de giros y alcanzando el estado menta fijado por el usuario.

Capítulo 6. Planificación del trabajo y Entorno Socio-Económico

En este capítulo, se realiza una descripción de la metodología utilizada para el desarrollo del proyecto. Esta metodología, describe las pautas a seguir para que el proyecto tenga éxito. A partir de estas pautas, se realizará una estimación del tiempo y los recursos a utilizar en cada una de ellas. Esto proporcionará una mayor adecuación y facilitará el proceso de creación del proyecto. Basándose en las pautas y los recursos a utilizar, se realizará el presupuesto del proyecto.

6.1 Metodología Empleada

En este proyecto se utilizará para el seguimiento y control un ciclo de vida en cascada. Se elige este modelo porque es la más conveniente para el cumplimiento de los requisitos descritos en el capítulo de Análisis del Sistema, en el apartado 3.4. Este modelo fue propuesto por Royce 1970 (9), y fue adaptado a la ingeniería del software a partir de ciclos de vida de otras ramas de la ingeniería. Este modelo realiza iteraciones en sus partes para poder hacer cambios en fases que ya han sido concluidas y que se ha detectado que eran incorrectas. En la *Ilustración 6.1-1* se muestra la iteración de cada una de las fases del modelo en cascada utilizada en este proyecto.

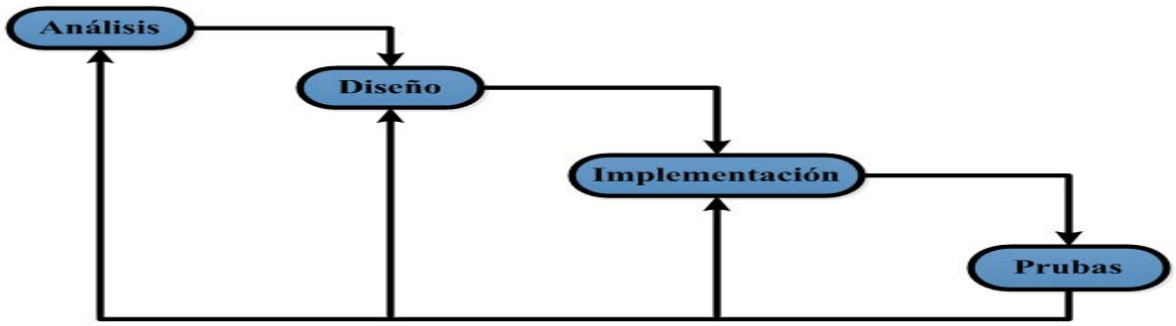

Ilustración 6.1-1. Metodología en Cascada.

La descripción de las partes de esta metodología es la siguiente:

- **Análisis:** Esta fase consiste en la realización de un análisis de todos los objetivos que debe cumplir la aplicación. De esta fase, sale un listado de requisitos que debe desempeñar la aplicación. En esta fase, cabe destacar que se realizará un listado de las necesidades de software y hardware que se requieren para implementar la aplicación.

- **Diseño:** Consiste en la descripción de la estructura del sistema y la descripción de lo que harán cada una de sus partes. También tendrá descrito la manera en la que se dispone con

las demás partes. En esta parte conviene intentar separar las partes a implementar de la manera que resulte más eficiente a la hora de desarrollar. Esto es debido, a que si se tiene un equipo de desarrollo, resultará más eficiente implementar la aplicación.

- **Implementación:** En esta fase se realiza la implementación del sistema diseñado en la fase anterior, con el fin de facilitar el proceso de implementación. Esta parte se basará en la creación de los componentes diseñados en la fase anterior. Se implementará componente por componente y después se ensamblarán todos.

- **Pruebas**: Se comprueba que se cumple con todos los requisitos especificados. En esta fase del proyecto, es donde pueden surgir más problemas, por lo cual, es donde pueden aparecer más iteraciones en el ciclo de vida del proyecto. Aunque se pueden producir varias iteraciones por culpa de los errores, estos deberían subsanarse rápidamente, debido a que el equipo en esta fase está lo instruido en el tema del proyecto.

En cada una de estas fases, intervendrán uno o varios miembros del equipo de trabajo. Estos realizarán diferentes tareas para lograr los objetivos propuestos para este proyecto.

6.2 Roles y Atribuciones

En este apartado, se definirá el equipo que se encargará de desarrollar la aplicación, diseñándola en un primer lugar y posteriormente programándola. El equipo está formado por un analista, un documentalista y un desarrollador. Las atribuciones de estos roles son las descritas en el (BOE. Capítulo III., Miércoles 10 de Octubre 2012.) (23). En los siguientes puntos se realizará una descripción más profunda de cada uno de estos roles:

- **Analista:** La responsabilidad del Analista es elaborar un catálogo detallado de requisitos que permita describir con precisión el sistema de información. Para ello, mantendrán entrevistas y sesiones de trabajo con los responsables de la organización y usuarios, actuando de interlocutor entre estos y el equipo de proyecto, en lo que a requerimientos se refiere. También, se encarga del diseño de la aplicación para que el desarrollador pueda llevar a cabo su función.

- **Desarrollador:** La función del desarrollador, es construir el código que dará lugar al producto resultante, en base al diseño técnico realizado por el analista, así como generar el código asociado a los procedimientos de migración.

- **Documentalista:** La función del Documentalista es de recompilar toda la información necesaria para el proyecto y plasmarla en los diferentes documentos. También, se encargará de catalogar dicha información.

Para que estos miembros del equipo puedan realizar sus tareas, necesitarán disponer de recursos. Estos recursos puede que no estén disponibles, debido a que estén siendo utilizados en otra tarea, haciendo que no puedan ser utilizados por alguno de los miembros del equipo. Por eso, será necesario realizar una buena planificación de éstos.

6.3 Planificación de actividades y recursos

En este apartado, se realizará una planificación temporal de las actividades y los recursos disponibles para el desarrollo del proyecto. Esta planificación ayudará a que se cumplan los objetivos del proyecto antes de la fecha de entrega. Para la realización de esta planificación, se tendrá encuentra la dificultada de la tarea a desarrollar, así como la disponibilidad de los recursos para el desarrollo de ésta.

6.3.1 Seguimiento y Control

El proceso de seguimiento y control de un proyecto, es una de las actividades más importantes en el desarrollo del mismo, ya que un correcto seguimiento y control, podrá evitar desviaciones en los plazos y por lo tanto en el gasto. Por ello, se deberá prestar atención a desarrollar un plan de seguimiento y control adecuado.

Para realizar la planificación y seguimiento del proyecto, se deberá utilizar herramientas que cumplan las siguientes características:

- Que sea fácil de utilizar para todo el equipo de trabajo.
- Que sea útil (aporte datos relevantes).
- Que sea gratuita (freeware).

En las siguientes partes, se ve como se ha distribuido las actividades y los recursos a lo largo del tiempo planificado. La herramienta utilizada ha sido GANTT PROYECT (24). Esta herramienta proporciona un diagrama Gantt y un diagrama de recursos.

Un diagrama de Gantt, es una herramienta gráfica que se utiliza tradicionalmente en proyectos que están expuestos a incidencias y modificaciones, puesto que se pueden readaptar fácilmente. En la *Tabla 6.3-1,* se muestra los ítems para el desarrollo de las diferentes actividades del proyecto.

Tasks			
Nombre	**Fecha Inicio**	**Fecha Fin**	**Duración en Días**
Análisis	9/04/12	23/05/12	33
Diseño	14/05/12	3/09/12	81
Implementación	18/06/12	19/12/12	133
Pruebas	5/12/12	9/01/13	26

Tabla 6.3-1. Task, Seguimiento y Control.

La distribución de las actividades, se muestra en la *Ilustración 6.3-1* que pertenece al diagrama de Gantt de este proyecto.

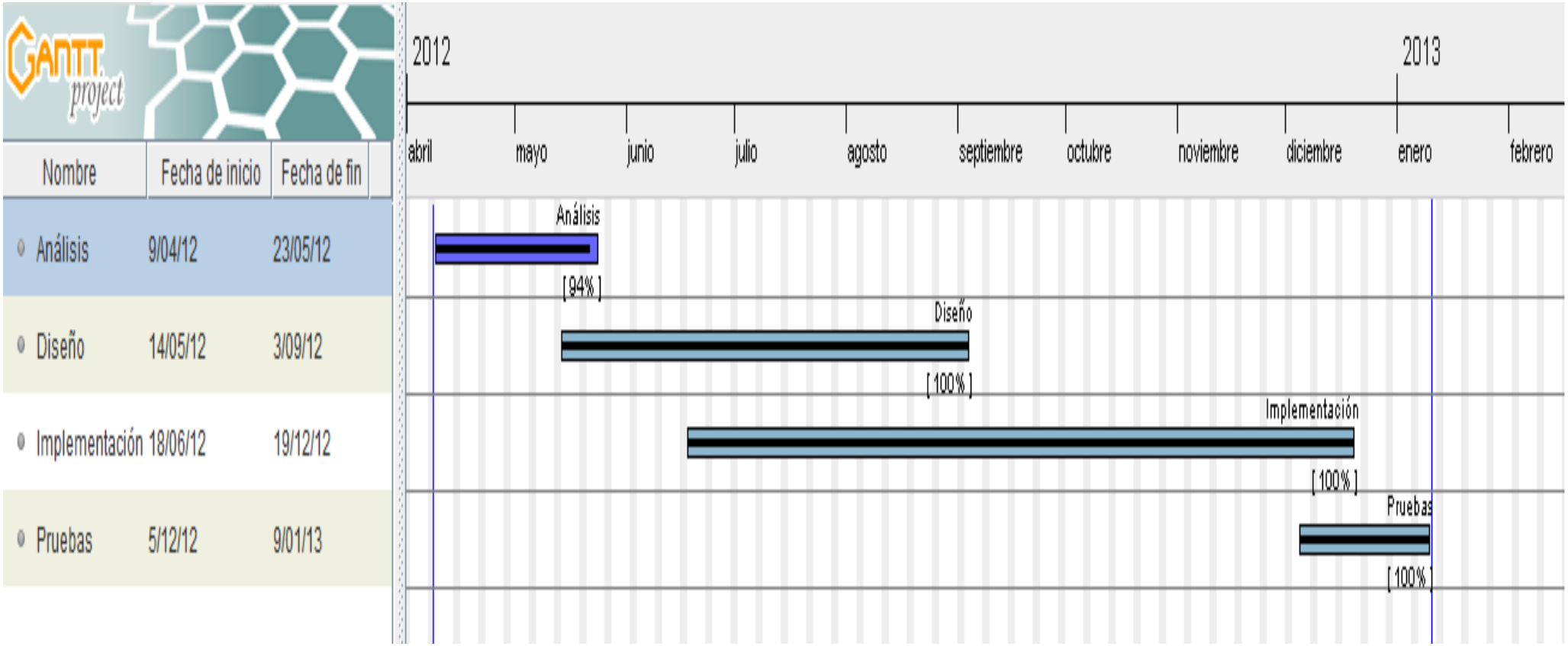

Ilustración 6.3-1. Diagrama de Gantt.

6.3.2 Asignación de Tiempo a Recursos

En este apartado, se realizará una estimación de los recursos que se necesitan para lleva a cabo este proyecto y como se distribuirán a lo largo del tiempo, previsto en el apartado anterior. Los recursos de los que se dispone para hacer este proyecto son los siguientes:

- **Recursos Humanos**: Representan a las personas que serán necesarias para el desarrollo del proyectos. Se dispondrán de tres personas, cada una con los siguientes roles que son los explicados en el apartado 6.2.

    - Analista.
    - Documentalista.
    - Desarrollador.

- **Recursos Materiales**: Los materiales utilizados para el desarrollo del proyecto son los siguientes.

    - Robot P3DX: La descripción de este recurso está explicado en el Anexo IV. Hardware del Sistema.

    - Cámara Microsoft Kinect: La descripción de este recurso está explicado en el Anexo IV. Hardware del Sistema.

    - Computador ASUS A53S: Este Computador consta de las siguientes características:

        - Procesador Intel(R) Core™ i7 2630QM.
        - Tarjeta Gráfica. Nvidia Geforce GT 540M, 1GB DDR3 VRAM.
        - 4GB de DDR· 1333MHZ DRAM, 2 slots de 8GB máximos.
        - Puertos HDMI y VGC, Un puerto Wireless 802.11 bgn, un Puerto USB 3.0 y 2 puertos USB 2.0.

En la *Ilustración 6.3-2,* se muestra el diagrama de la distribución estimada de los recursos humanos y materiales a lo largo del periodo estimado en el apartado anterior. Este gráfico será de gran utilidad para la creación del presupuesto del apartado 6.4.

Ilustración 6.3-2. Diagrama de Asignación de Recursos.

Este diagrama muestra los recursos utilizados en cada fase de la metodología. Cada cuadrado del diagrama representa una semana, los cuadrados en rojo representa la sobreasignación de recursos. Una sobreasignación de recursos se debe a que hay una solapación de fases en el tiempo, y cada una de estas fases comparten recursos, por lo cual, en el mismo tiempo un recurso puede realizar dos tareas. Esto no debería ser bueno, porque puede haber saturación de recurso y disminuir el rendimiento de éstos. En este proyecto, se llega a tolerar esta sobreasignación, ya que la saturación es admisible. Esto se debe a que los recursos disponibles tiene la capacidad de realizar dos tareas a la vez. Si el recurso realiza más de dos tareas a la vez habría que realizar una replanificación.

6.4 Presupuesto del Proyecto

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

1.- Autor:
Félix Caro Herranz.
2.- Departamento:
Departamento de Informática.
3.- Descripción del Proyecto:
- Título: Sistema de Planificación para el Robot P3DX.
- Duración (meses): 9
- Tasa de costes Indirectos: 21%
4.- Presupuesto total del Proyecto (valores en Euros):
35.780,00 €
5.- Desglose presupuestario (costes directos):

PERSONAL				
N.I.F.	**Categoría**	**Dedicación de meses por hombre**	**Coste hombre mes[1]**	**Coste (Euro)**
70070776B	Desarrollador	8	1302,52	10.420,16
70070776B	Documentalista	5	1320,67	6.603,35
70070776B	Analista	4,3	1574,83	6.771,77
			Total	**23.795,28**

[1]Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas). Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas). Sueldos Base según el BOE a fecha de 7 de Marzo de 2012 (25).

EQUIPOS					
Descripción	**Coste (Euro)**	**%Uso dedicado proyecto**	**Dedicación (meses)**	**Periodo de depreciación**	**Coste imputable [d)]**
Ordenador Intel(R) Core™ i7	850	50	9	60	63,75
Robot Pionner-3DX	2568,5	78	8	60	267,12
Cámara Microsoft Kinect	121,391	78	8	60	12,62
				Total	**343,50**

$\frac{A}{B}$xCxD

[d)] Fórmula de cálculo de la Amortización:
A = nº de meses desde la fecha de facturación en que el equipo es utilizado.
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS		
Descripción	**Empresa**	**Coste imputable**
Instalación Kinect	Departamento de Mecánica	50,00
	Total	**50,00**

OTROS COSTES DIRECTOS DEL PROYECTO[e)]		
Descripción	**Empresa**	**Costes imputable**
Internet	Movistar	60,00
Microsoft Office 2007	Microsoft	130,00
Luz	Endesa	60,00
	Total	**250,00**

[e)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros.

6.- Resumen de costes directos

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	23.795
Amortización	343
Subcontratación de tareas	50
Costes de funcionamiento	250
Costes Indirectos	5.132
Total	**29.571**

Capítulo 7. Conclusiones y Líneas de Trabajo Futuras

En este capítulo del documento, se exponen las conclusiones obtenidas tras la realización del proyecto. Se realizará una descripción de las conclusiones por objetivos para hondar más en el tema y poder realizar una descripción lo más completa posible. Se describirá las posibles líneas de trabajo que se pueden realizar, basándose en el sistema de este proyecto. Por último, se describirán los problemas surgido en el ciclo de vida del proyecto y como se resuelven.

7.1 Conclusiones Generales

En este apartado, se plasma las conclusiones generales que se han obtenido al realizar este proyecto. Después de este apartado, se realizará una descripción más exhaustiva de las conclusiones basándose en los objetivos a cumplir.

Los waypoints se crean cuando se efectúa el proceso de generación del mapa. Esto hace que sea más eficiente el añadir waypoints en los mapas, ya que a la vez que se crea el mapa, se realiza el proceso de creación de waypoints, por lo tanto, se ejecutan dos tareas al mismo tiempo. Otra ventaja de formalizarlo de esta manera, es que supone una mayor adaptación entre los waypoints creados y los parámetros del entorno que recibe el robot, por lo cual, el error que pueda surgir en la navegación se minimiza.

Otra conclusión que se ha obtenido sobre este proyecto, son los errores que surgen en el proceso de coordinación de los componentes del software de un robot. Esto se debe a que el robot actúa en tiempo real y por lo tanto, este sistema tiene que ser tolerante a fallos. Este es un problema muy difícil de tratar, debido a que cuando sucede un fallo, se produce una demora en el cálculo, haciendo que el proceso de creación de waypoints no concuerde con la realidad que captan los sensores. Otro contratiempo que ha surgido en la realización de este proyecto, es la inadecuación de los datos percibidos del entorno, debido a que este es dinámico. Por lo tanto, habría que realizar una reconstrucción de la percepción del entorno en tiempo real, produciéndose una pérdida de tiempo.

El sistema implementado, es un sistema de tiempo real[6] y por lo tanto, implican unas restricciones en el tiempo, pues deben adaptar a la evolución temporal del entorno físico que controlan y que impone sus límites de tiempo sin admitir demoras.

[6] Sistema de Tiempo Real. Las aplicaciones de tiempo real son las aplicaciones informáticas en las que la obtención de los resultados está sujeta a unas restricciones temporales impuestas por el entorno en que se ejecutan.

Este sistema esta implementado de manera que se puedan realizar actualizaciones sobre él, lo cual, supone que puedan aparecer problemas de mantenimiento, debido a que cualquier cambio requiere una nueva verificación detallada para asegurar la vigor del comportamiento, tanto funcional como temporal, pues la modificación de un determinado proceso puede afectar al comportamiento temporal del resto. Por eso, cuando se realiza una actualización del sistema se debe comprobar que las funcionalidades de éste no se ven afectadas.

.

7.2 Conclusiones por Objetivos

En este apartado se realizará la descripción de las conclusiones de los objetivos iniciales propuestos para este proyecto. Se explicará si se han alcanzado todos los objetivos propuestos, que datos de interés se ha obtenido y como podría mejorarse ciertos procesos. Las conclusiones de los objetivos son las siguientes:

- **Exploración por el entorno:** Este objetivo se ha cumplido correctamente en todas las pruebas realizadas. El robot es capaz de alcanzar todas las zonas de un entorno y esquivar los obstáculos que se pueda encontrar. Como se puede observar en las pruebas, el robot se maneja muy bien en entornos amplios. Sin embargo, cuando explora entornos cerrados y pequeños tarda bastante, ya que el sonar recibe muchos datos y tarda en procesarlos. Esto no es del todo un inconveniente, ya que moviéndose más despacio en estos entornos, será más fácil recrearlos y obtener un grado mayor de fiabilidad en la posición de los objetos. Uno de los problemas que ha surgido aquí, es que el robot en muchos casos no vuelve al lugar donde empezó a explorar. Esto supone un problema para la creación del mapa, ya que los objetos no se representa de forma precisa. Este problema solo surge en entornos abiertos, debido a que el robot se desplaza por ellos sin encontrar un final y haciendo que este nunca vuelva al lugar donde partió. Para representar bien un entorno es conveniente que el robot de varias pasadas para definir bien los objetos de éste.

- **Creación del mapa:** Los mapas se crean bastante bien, mostrando todos las marcas topológicas del entorno. Cabe destacar, que se necesita volver al lugar de partida de la exploración, para obtener una mayor definición de las marcas topológicas. Como se ha comentado en el punto anterior, esto no sucede en lugares abiertos, solo ocurre en entornos cerrados.

- **Creación de Waypoints:** Los Waypoints se crean perfectamente en las zonas libres, y la descripción del eje de coordenadas coincide con las zonas libres de obstáculos del mapa. Sin embargo, las conexiones entre waypoints cuando el tamaño de la rejilla es grande, es incorrecto, esto es debido a que es imposible conectar determinadas zonas en las que hay demasiados obstáculos. Esto no es del todo un problema, ya que como el tamaño de la rejilla puede ser definido por el usuario basta con reducirlo para que se produzca la conexión entre todos los waypoints. Además, esto servirá para advertir de que no se podrán crear rutas que alcancen todo los puntos de un entorno para robots grandes.

- **Navegación autónoma por el entorno:** La navegación por entornos se realiza de manera correcta en todas las pruebas. Alcanzando los puntos meta con la mayor eficiencia en movimientos posible. Me gustaría recalcar en este punto, que se debería utilizar el sonar para evitar obstáculos imprevistos que no se recogen en el mapa. Aunque el infrarrojo de la Kinect los capta bastante bien, tarda un poco en procesar la información, haciendo que incremente el riesgo de colisión aumente. Por eso, se debería implementar una funcionalidad con el sonar para complementar la navegación con el infrarrojo y obtener una mayor seguridad a la hora de percibir los obstáculos del mapa.

7.3 Líneas de Trabajo Futuras

En este apartado del capítulo, se explican los posibles trabajos que se pueden realizar con el software y hardware utilizados en este proyecto. Se realizará una descripción del problema y un análisis de las posibles soluciones a tomar.

Debido a que en este proyecto se ven varias partes de la robótica móvil, se podría seguir trabajando en diferentes temas como son la planificación y la navegación de un robot móvil. Las posibles líneas de trabajo son las siguientes:

- Un proyecto a desarrollar, sería crear un sistema colaborativo con el robot NAO que es el que aparece en la *Ilustración 7.3-1*. Basándose en la creación de waypoints de este proyecto, se podría realizar un sistema en el que un planificador automático planificase un plan y se lo pasase al robot NAO. Este plan, sería recorrer una serie de waypoints para llegar a una meta. El robot Nao es el que llevaría a cabo este plan. Para obtener los waypoints, habría que conectarse a nuestro sistema a través de la interfaz mensaje_waypoint, descrita en el apartado Arquitectura del Sistema, del capítulo de Diseño de la Solución.

Ilustración 7.3-1. Robot NAO.

El sistema implementado es muy apropiado, ya que permite al robot NAO, encontrar sus objetivos en un mapa que pueda reconocer su sistema. Este sistema daría solución al problema de creación de mapas para un sistema basado en el robot NAO. El robot NAO no tiene la capacidad de realizar mapas de un entorno, por eso este sistema le ayudaría a resolverlo.

- Con respecto a proyectos que abraquen el problema de planificación, se podría realizar la implementación de un robot camarero. Este robot tendría que llevar las bebidas a distintas mesas de un local. La explicación de la aplicación sería la siguiente:

  El robot tendría que moverse por rutas planificadas que le llevasen a las mesas en el menor tiempo posible. Por el camino se podría encontrar con obstáculos que deben ser esquivados. Además, este robot tendría un registro de las mesas que están ocupadas y están vacías. Para a aproximarlo al problema de planificación la salida sería la barra donde están las bebidas y la meta la mesa a servir. La *Ilustración 7.3-2* muestra un posible entorno de trabajo para esta aplicación.

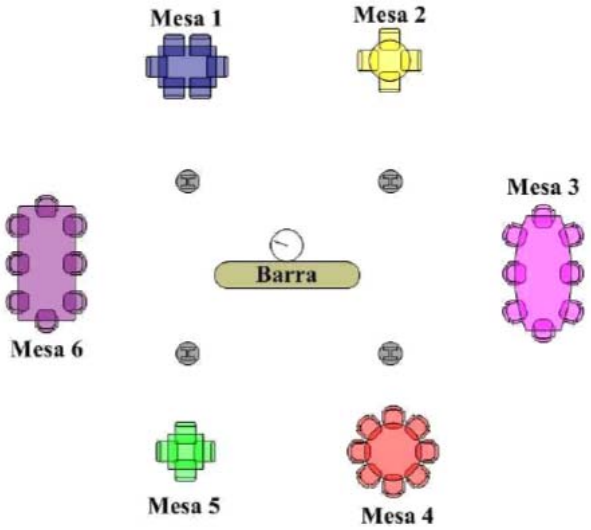

Ilustración 7.3-2. Distribución de mesas.

En esta aplicación, se podría utilizar casi en su totalidad la implantación de este proyecto. La única diferencia que existiría, sería la creación de waypoints con un estado que describiese si la mesa está ocupada o vacía.

Como se puede observar, existen varios proyectos que se pueden realizar basándose en el proyecto desarrollado.

7.4 Problemas Encontrados

En este apartado se realizará una descripción de los problemas que se han encontrado durante el desarrollo de este proyecto. El objetivo de este apartado, es presentar cada uno de los errores y mostrar la forma de evitarlos o sortearlos.

Los siguientes puntos muestran los diferentes errores que han aparecido en este proyecto:

1. **Creación de la rejilla:** Aunque en los experimentos realizados no aparecido este problema, si se ha observado un error a la hora de establecer waypoints en lugares cercanos a objetos. Este error, se debe a la conexión de entre waypoint de celdas en las esquinas de una celda libre. Para una mayor comprensión de esta explicación, véase la *Ilustración 7.4-1*.

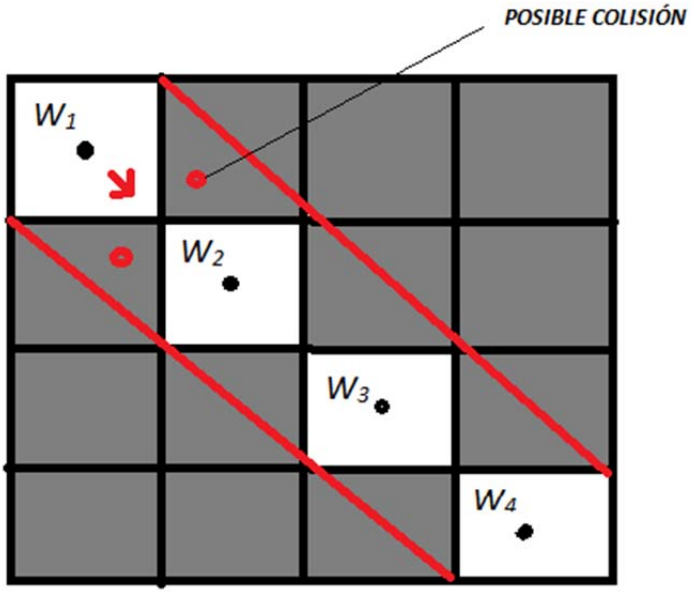

Ilustración 7.4-1.Problema de Colisión.

Si se quiere realizar un desplazamiento desde el waypoint W_1 hasta W_2, podría haber una posible colisión debido a que se desconoce lo que hay en las casillas adyacentes a la del W_1. Para evitar este error, lo que se debe de hacer es un análisis de los pixeles adyacentes al waypoint W_1. Véase la *Ilustración 7.4-2*

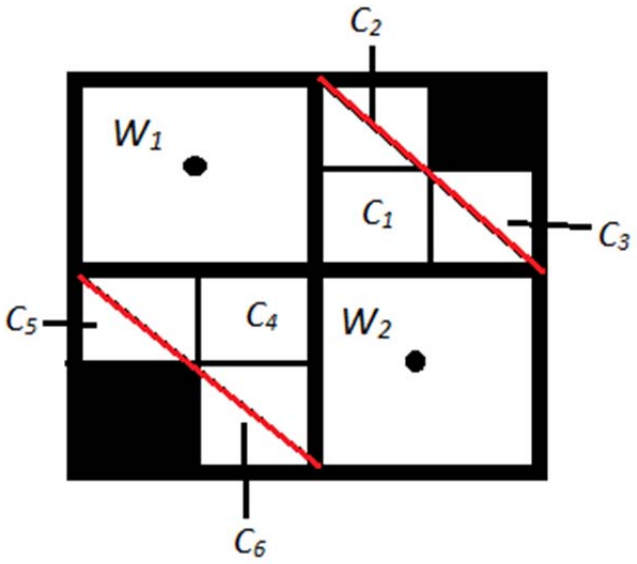

Ilustración 7.4-2. Solución rejilla.

Para comprobar que se puede establecer una conexión desde el waypoint W_1 hasta el waypoint W_2, habría que hacer una división de las casillas adyacentes en cuatro subcasillas como muestra la *Ilustración 7.4-2*. Entonces, se comprobaría que las subcasillas C_1, C_2, C_3, C_4, C_5, C6, están libres de obstáculos. En caso de que este libres de obstáculos, se podría realizar una conexión. Este error no ha surgido en la experimentación debido a las siguientes causas:

- Como se puede apreciar en los experimentos, en los lugares donde ha sido marcado con un waypoint, las casillas adyacentes tiene libre de obstáculos las partes que representarían las subcasillas de la *Ilustración 7.4-2.*

- Otra causa por la que no ha sucedido una posible colisión, es por el inflado de obstáculos que realiza el componente Move_base, descrito en la arquitectura del sistema.

2. **Versiones ROS:** ROS tiene cuatro versiones que son las siguientes:

- 23/Abril/2012 – Fuerte
- 30/Agosto/2011 - Electric Emys
- 02/Marzo/2011 - Diamondback
- 03/Agosto/2010 - C Turtle
- 01/Marzo/2010 - Box Turtle.
- 22/Enero/2010 - ROS 1.0

Para este proyecto, se ha utilizado la versión Diamondback. Esta versión ofrece todas las características para poder realizar el desarrollo de este proyecto. Cabe destacar, que las versiones de Fuerte y Electric Emys contienen paquetes muy útiles, pero que no son compatibles para realizar las funcionalidades que se han llevado a cabo. Otro gran problema, es que se han utilizado paquetes que no tenían una versión estable en las versiones más modernas de ROS.

Por último, la documentación ha cambiado de referencia en ROS, al igual que algunos repositorios. Si usted necesita instalar algún stacks de ROS para Diamondback, puede que no lo encuentre, busque en otros sitios que no sea la página oficial de ROS, porque es muy probable que lo halle.

3. **Conexión USB:** Los puertos USB de conexión con el robot y la cámara Kinect tiene que ser USB2.0, ni superior ni inferior, ya que los buffers de entrada de los controladores se queda sin memoria, lo que hará que estos generen mensajes de información de los dispositivos y haciendo que el sistema falle. Este error, se puede identificar si la consola de ROS muestra el Warning:

”100% Dropped messagge [/openni_kinect, /slam, /p2os_driver, /RosAria]”.

Este error hizo que se perdiera mucho tiempo en la implementación del sistema, ya que se creyó en un principio que el error surgía de un problema de asignación de memoria de ciertas variables en los componentes Mapa y Waypoints.

4. **Entorno RVIZ:** La interfaz de visualización y monitorización que ha sido utilizado, necesita de una tarjeta gráfica compatible con el soporte Microsoft DirectX® 9.0c**.** En un principio no se tuvo en cuenta esto y origino que no se pudiese realizar una monitorización y visualización de los proceso de creación de mapas y navegación**.** Ubuntu 11.04 ofrece uno muy parecido ha permitido el desarrollo de este proyecto. Pero hay que tener muchísimo cuidado porque las actualizaciones de los controladores de la tarjeta gráfica de este sistema hacen que estos sean incompatibles. Por lo tanto, como este sistema operativo ofrece la posibilidad de actualización de ciertos paquetes, se debe descartar la opción de actualización de los controladores de la tarjeta gráfica. Este error ha hecho que se pierda, gran cantidad de tiempo en la implementación del sistema

5. **Memoria Física:** Se debe garantizar que para la ejecución de la aplicación, se tiene al menos 1.5 GB de memoria, sino se producirá pérdida en los mensajes de los nodos de ROS. Este error, surgió en el proceso de creación de mapas, ya que la hora de establecer el estado de los pixeles no se disponía de la suficiente memoria

6. **Instalación Openni_Kinect:** En la documentación de ROS para la instalación del stacks “al menos en la antigua” no advierte de que el usuario debe estar registrado en el grupo video del sistema operativo, ni que se necesita tener instalada la librería *openjdk-6.* Esto hace que algunas funcionalidades de la cámara queden inservibles. Cuando instale este paquete, no realicé las comprobaciones que viene en la documentación de ROS y el infrarrojo de Kinect no funcionaba, pero si la cámara. Además, si no se tiene en cuenta esto, el sistema no mostrará ningún mensaje de error y será muy difícil encontrar el fallo de la funcionalidad deseada.

7. **Máquinas Virtuales:** La implementación de este proyecto en algunas máquinas virtuales puede suponer algunos problemas, sobre todo con las conexiones de los dispositivos del robot y de la Kinect, ya que hay pérdida de funcionalidades. Por eso, desde mi punto de vista, es bueno realizarlo directamente con una máquina física. Si usted utiliza copias de seguridad de su sistema, no tiene por qué tener muchas complicaciones al hacerlo de esta manera.

ANEXO I. Manual de Instalación

Este manual de instalación define como realizar de manera satisfactoria todas las actividades descritas anteriormente para poder instalar la aplicación desarrollada. Este Manual de Instalación tiene por objetivo que la implantación se lleve a cabo, facilitando al usuario su labor. El proceso de la implantación tiene por objeto alcanzar los siguientes objetivos concretos:

- Realizar un análisis del estado del software de los componentes del equipo donde se realizará la instalación.
- Instalar el software necesario para realizar la implantación.
- Verificar y comprobar que el software se ha instalado correctamente.

8.1 Preparativos de Implantación.

En este apartado, se definen las diferentes fases de la implantación que el usuario tendrá que cumplir. Este apartado consta de una serie de comprobaciones que hay que realizar para poder realizar la instalación. Las comprobaciones se dividen en dos, las comprobaciones de hardware y las de software.

Comprobaciones del Hardware:

1. **Comprobación de conexión a internet:** Se necesita tener conexión a internet para la descarga de los paquetes de ROS**.**
2. **Comprobar la tarjeta Gráfica:** Tarjeta gráfica que soporte Microsoft DirectX® 9.0c**.**
3. **Espacio del disco:** Tiene que haber al menos 4GB de espacio libre en el disco duro.
4. **El sensor de Kinect para Xbox360:** Las aplicaciones desarrolladas con el SDK de Kinect deben ejecutarse en máquinas con Linux nativo, **no en máquinas virtuales**, porque el *driver* PCKinect y la SDK deben estar instaladas en el ordenador donde se ejecuta la aplicación.
5. **Procesado**r: Doble núcleo, 2.66GHz o superior.
6. **Memoria:** 2 GB de RAM.

Comprobaciones del Software:

1. **Sistema operativo:** El sistema operativo no debe ser una versión superior a 11.04 Ubuntu Natty**.**
2. **Librerías del sistema operativo:** El sistema operativo debe de tener instalado las librerías *mercurial, core-git, openjdk-6, python-setuptools, easy_install*.
3. NO Tener las **últimas actualizaciones** Ubuntu 11.04 Natty.
4. **No tener conectado** el dispositivo Kinect al puerto *USB* del ordenador.
5. **Eliminar** cualquier otro ***driver*** de PCKinect que tengas previamente instalado. Podría no instalarse correctamente y no funcionar.
6. **Dar permisos** de ejecución, escritura y lectura a la carpeta de *workspace* para comprobarlos permisos hacer lo siguiente**:**

```
-rvxr—x--- 1 pepit depart1 4348 Nov 24: 16:19 test
```

Para añadir permisos realizar lo siguiente**:**

```
chmod 777 -R nombreCarpeta
```

8.2 Fases de la Implantación.

El usuario deberá instalar paso por paso el software necesario para que todo funcione correctamente en su computadora. A medida que se vaya realizando la instalación, se podrán ir haciendo comprobaciones debido a que se instala por módulos. Los pasos para realizar la instalación son los siguientes:

1. Abrir una terminal y ejecutar los siguientes instrucciones:
   Instalar ROS Diamondback.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu natty main" > /etc/apt/sources.list.d/ros-latest.list'
```

```
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

```
sudo apt-get update
```

```
sudo apt-get install ros-diamondback-desktop-full
```

```
echo "source /opt/ros/diamondback/setup.bash" >> ~/.bashrc
. ~/.bashrc
```

2. Crear un espacio de trabajo y hacer dependencias con los paquetes de ROS.

```
rosws init ~/workspace /opt/ros/diamondback
```

```
gedit  ~/.bashrc
```

Se abrirá un editor de textos y al final del archivo se escribirá lo siguiente:

```
export ROS_ROOT=/opt/ros/diamondback/ros
export PATH=$ROS_ROOT/bin:$PATH
export PYTHONPATH=$ROS_ROOT/core/roslib/src:$PYTHONPATH
export ROS_PACKAGE_PATH=~/workspace:/opt/ros/diamondback
/stacks:$ROS_PACKAGE_PATH
```

3. Crear un paquete en el espacio de trabajo.

```
 cd ~/workspace
```

```
$ roscreate-pkg proyecto
```

Copiar las carpetas del proyecto a este proyecto.

4. Descargare instala los paquetes Openni_kinect, amor.

```
sudo apt-get install ros-diamondback-openni-kinect
```

Descarga de la librería amor:

```
sudo hg clone https://amor-ros-pkg.googlecode.com/hg/
     local-name: Pionner/amor-ros-pkg
```

5. Compilar el paquete del proyecto.

```
rosmake proyecto
```

Para añadir alguna dependencia de ROS, se hace lo siguiente:

```
$ roscd proyecto
$ cat manifest.xml
```

Se introduce en package el nombre de la dependencia al paquete.

```
rosdep install [package]
```

6. Conectar al ordenador la cámara Microsoft Kinect y el y el cable USB al robot.
   Se conecta el cable USB de la Microsoft Kinect a la computadora donde se va a realizar la ejecución de la ampliación. Por otro lado, se conecta el cable de alimentación de la cámara Microsoft Kinect a una fuente de alimentación.
   Para conectar la computadora al robot se utiliza un cable

7. Encender el robot. En esta parte se da al botón de conexión del motor que muestra la figura siguiente

Ilustración 8.2-1. Toma de conexión Robot P3DX.

Una vez realizadas las conexiones, para probar que éstas se han establecido sin problemas, se realizará la siguiente instrucción.

```
roscore
```

En otro terminal:

```
rosrun p2os_dashboard p2os_dashboard
```

En caso de que se muestre algún erro habrá que repetir los procesos 6 y 7.

8. Lanzar la aplicación. Para lanzar la aplicación basta con ejecutar la siguiente instrucción.

```
roslaunch proyecto proyecto.launch
```

Una vez realizado esto, el robot empezará a desplazarse por el entorno. Si se quiere para este proceso basta con pulsar Ctrl+C en la terminal donde se ejecuta la anterior acción y el proceso terminará.

ANEXO II. Manual de Usuario

En este apartado se realizará la descripción del funcionamiento de la aplicación.

Cuando se quiere realizar la exploración de un entorno y crear una imagen del entorno y realizar la creación del fichero XML con la información de los waypoints se ejecuta la siguiente instrucción en el terminal de Linux

```
roslaunch proyecto crear mapa.launch
```

Para la correcta realización del mapa, será necesario dar varias pasadas por el entorno que se quiere realizar el mapa, saliendo de un punto de partida y regresando a él para realizar mejor el bucle de obtención de datos.

Para poder visualizar el proceso de creación del mapa, se lanzará el Rviz, que como se cometa en el Anexo III ROS, es la herramienta de ROS que permite la visualización y monitorización de procesos. Para lanzar Rviz se utiliza el siguiente comando.

```
rosrun rviz rviz
```

Una vez que se cree el mapa, para que el robot se desplace por él, se hará uso de la librería de ROS "navigation", que tiene los paquetes "map_server", "amcl" (adaptive Monte Carlo localization) y “move_base”. Cada uno de ellos, respectivamente, carga el mapa creado, ejecuta el método probabilístico de localización adaptativa Monte Carlo y planifica la ruta para ir del origen al destino dentro del mapa evitando los obstáculos. El *package* “move_base” se encarga de la planificación de rutas, navegación global y de evitar de obstáculos a través de la navegación local.

Para poder obtener un fichero con los datos del mapa, será necesario ejecutar el archivo del proyecto “navegation.launch”. Se utilizará otra vez el programa Rviz para planificar la ruta que ha de seguir el robot. Las instrucciones del terminal para ejecutar estas acciones son las siguientes:

```
roslaunch proyecto navegation.launch
```

Para navegar en el entorno con el mapa realizado se deberá indicar la posición y orientación aproximada del robot dentro del mapa, pulsando en el botón "2D Pose Estimate" y posteriormente pulsando sobre el mapa en la posición estimada que se cree que esta el robot, sin soltar el botón derecho del ratón se indicará la posición que deberá alcanzar el robot. El modelo

del robot se situará en la posición revelada, rodeado por una serie de flechas rojas como se muestra en la imagen:

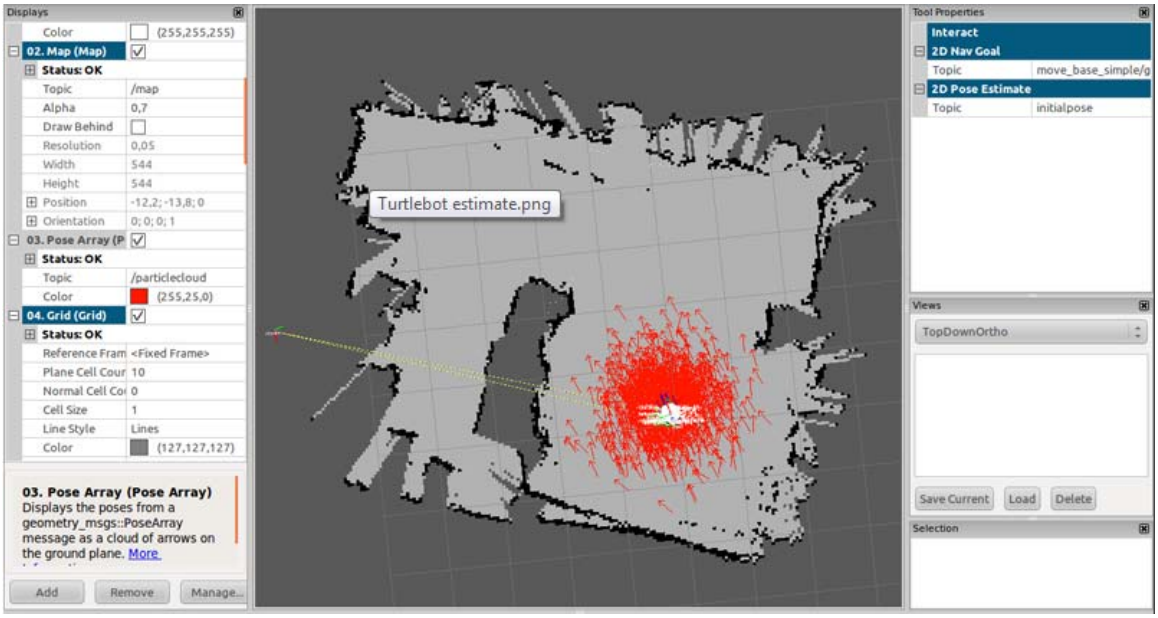

Ilustración 8.2-2. Navegación con RVIZ.

El proyecto consta de un archivo nav_rviz.vcg que contiene la configuración de parámetros de rviz para la correcta navegación del robot (19). Para abrir este archivo solo es necesario acceder a la venta File y después a la opción Load, aquí se indica el fichero a cargar, que es el descrito anteriormente.

ANEXO III. ROS

Este capítulo está basado en la plataforma ROS, en él se explicará los conceptos y funcionalidades y se describirá el modo en el que han sido integradas en el proyecto. Con ello se pretende que los lectores adquieran conocimiento de la herramienta utilizada para el desarrollo del proyecto.

10.1 Introducción a ROS

ROS (System operation Robots) fue creado 2007 en por el departamento de inteligencia artificial de la Universidad de Stanford. A partir del 2008 tomo relevo en el desarrollo el instituto de investigación Willow Garage. Como se ha comentado anteriormente el código es libre y abierto para uso y disfrute de todo el mundo. Lo que se pretende con esto es que este vaya mejorando y creciendo (19) (5).

ROS es un código abierto, meta-sistema operativo para robots. Ofrece los mismos servicios que un sistema operativo, incluida la de abstracción de hardware de bajo nivel de control del dispositivo, la aplicación de la funcionalidad de uso común, de paso de mensajes entre los procesos y de gestión de paquetes. También proporciona herramientas y bibliotecas para la obtención, la construcción, la escritura y la ejecución de código en varios equipos.

El tiempo de ejecución de ROS "gráfico" es una red *peer-to-peer* de los procesos que están débilmente acoplados utilizando la infraestructura de comunicación propia de ROS.

ROS implementa diferentes estilos de comunicación, incluyendo la sincronización de servicios RPC, la transmisión asíncrona de información a través de datos y topics almacenados en un servidor de parámetros.

Si los equipos de trabajo están conectados a una red heterogénea, no es necesario un servidor central para conseguir que todas las máquinas en servicio desarrollen los cálculos que requieren cada una de las tareas. La manera en la que se conectan lo muestra la *Ilustración 10.1-1*.

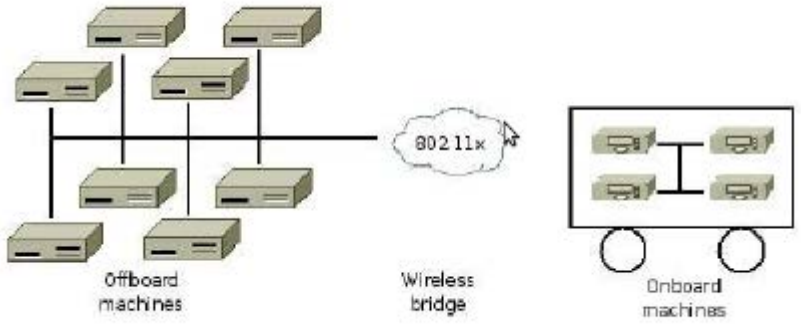

Ilustración 10.1-1. Infraestructura de Comunicación de ROS.

ROS también es una plataforma Multi-lenguaje, capaz de soportar lenguajes como C++, Python, LISP y Octave. En ROS están implementadas todas las funcionalidades en su forma nativa con estos lenguajes, en vez de implementarlas en C y crear una interfaz de código auxiliar para el resto de lenguajes. Con esto se consigue una mejor convección con los lenguajes. En este proyecto se ha utilizado el lenguaje C++ para el desarrollo de la aplicación. Además ROS permite implementar módulos en distinto lenguaje y conseguir que estos interactúen entre sí. Esto se consigue debido a que ROS tiene un lenguaje neutro en la interfaz de comunicación, este lenguaje se denomina IDL.

Para poder gestionar las operaciones entre módulos de manera eficiente ROS dispone de un kernel propio. Este kernel tiene varias herramientas que permiten realizar las siguientes operaciones de una manera más eficiente:

- Navegar por el código fuente.
- Obtener y establecer los parámetros de configuración.
- Medir la utilización del ancho de banda.
- Graficar datos de mensajes.

Se dispone de librerías que facilitan la reutilización de código de otros proyectos, cabe destacar que este es un proceso bastante complicado. ROS reutiliza código de otros proyectos como por ejemplo: simuladores del proyecto Player, algoritmos de visión de OpenCV, algoritmos de planificación de OpenRAVE, entre muchos otros. ROS es capaz de actualizar automáticamente código abierto de otros repositorios.
ROS distribuye código bajo la licencia BDS, que permite el desarrollo de proyectos comerciales como no comerciales. ROS pasa los datos entre módulos usando IPC (comunicación entre procesos) y no necesita que dichos módulos estén unidos al mismo ejecutable.

10.2 Conceptos fundamentales

En esta sección del capítulo se describirán los conceptos fundamentales de ROS. Lo que se pretende con ello es dotar a los usuarios de una visión más técnica del funcionamiento de este software.

10.2.1 Nodos

Un nodo es un proceso que realiza un cómputo. Los nodos se combinan en un gráfico y se comunican entre sí utilizando los topics de transmisión, servicios de RPC, y el servidor de parámetros. Estos nodos están destinados a operar a una escala de grano fino[7].

Un sistema (1) de control del robot comprenderá habitualmente muchos nodos. Por ejemplo, un nodo de control de un láser, un nodo control de los motores del robot, un nodo en cargado de realizar la localización, otro de la planificación de ruta, un nodo encargado de proporcionar una vista gráfica del sistema, etc.

El uso de nodos en ROS proporciona varios beneficios al sistema global. No hay tolerancia a errores, debido a que los nodos son independientes y la complejidad de código se reduce en comparación con los sistemas monolíticos[8].

Todos los nodos que se ejecutan tienen un nombre de recurso gráfico que los identifica unívocamente en el sistema. Por ejemplo, */ hokuyo_node* podría ser el nombre de un controlador de radiodifusión del Hokuyo láser. Los nodos pueden disponer de nodos superiores. Estos tipos de nodos son los que realizan la ejecución de todas las funcionalidades de un paquete, el nombre de este nodo adquiere el nombre del archivo ejecutable del paquete. Como tal, se necesita tener cuidado y no producir diferentes ejecutables con el mismo nombre en el mismo paquete.

ROS conecta los nodos de forma dinámica en tiempo de ejecución, las instrucciones que se utilizan a la hora de trabajar con nodos son las siguientes:

- **`rosnode`** = ros+node: Herramienta que proporciona información sobre un nodo.
- **`rosrun`** = ros+run : Ejecuta un nodo de un paquete determinado. En esta instrucción se puede hacer asignación de parámetros de la siguiente manera. **`Nombre_parametro:= valor`**, ejemplo `talker_param:=1.0`

10.2.2 Master

El Master de ROS es el principal nodo ejecutor de ROS, de él derivan todos los servicios y nodos, es el responsable de la comunicación entre estos.

El Master de ROS proporciona servicios de nombres que se ejecutan y el registro con el resto de los nodos del sistema de ROS. Realiza un seguimiento de los editores y suscriptores de los topics, así como de los servicios. El papel del Master permitir que los nodos individuales de ROS puedan localizarse unos a otros. Una vez que estos nodos se encuentran entre sí se comunican de igual a igual. El Master también proporciona el servidor de parámetros.

[7] Grano fino. Cuando una aplicación muestra sub-tareas y estas deben comunicarse muchas veces por segundo.
[8] Sistemas Monolíticos. El centro del software de este sistema está compuesto por estructuras fijas que interaccionan entre sí.

El Master se ejecuta comúnmente con el comando `roscore`, que carga el Master junto con otros componentes esenciales.

El Master de ROS proporciona una API XMLRPC basado en las bibliotecas ROS cliente, como roscpp y rospy, para llamar, almacenar y recuperar información. La mayoría de los usuarios de ROS no tendrá que interactuar con esta API directamente.

10.2.3 Mensajes

La comunicación entre los distintos nodos de ROS se realiza mediante mensajes. Están definidos en el directorio "*package-name/msg/*.msg*" y son enviados a través de los topics. Los tipos de datos básicos que utiliza son:

int {8,16,32,64}, float {32,64}, string, time, duration, array.

ROS utiliza un lenguaje de descripción de los mensajes simplificados para describir los valores de datos que los nodos publican. Esta descripción hace que sea fácil para las herramientas de ROS generar automáticamente código fuente para el tipo de mensaje en varios lenguajes de programación. Las descripciones de los mensajes se almacenan en archivos *.msg* en el subdirectorio */msg* de un paquete de ROS.

Hay dos partes en un archivo *.msg*; los campos y constantes. Los campos son los datos que se envían dentro del mensaje. Las constantes definen los valores útiles que pueden ser utilizados para interpretar esos campos (por ejemplo: de *enum-like* constantes para un valor entero).

10.2.4 Servicios

Los servicios son las funcionalidades que ofrecen los nodos. Permite enviarse solicitudes y respuestas unos a otros. Los servicios son de dos tipos:

- Subcritor: te inscribes a ese servicio con el fin de obtener la información que publica.
- Publicador: Publica información sobre el nodo.

10.2.5 Topics

Los topics son los nombres de los buses que los nodos usan en el intercambio de mensajes. En general, los nodos no son conscientes de que están comunicando información. Los nodos editan y se subscriben a un topics con la intención de trasmitir información. No puede haber varios editores y suscriptores a un mismo topics.

Los topics están destinados a la comunicación unidireccional. Los nodos que necesiten realizar llamadas a procedimientos remotos, es decir, recibir una respuesta a una solicitud, deben utilizar los servicios en su lugar. También existe el servidor de parámetros para el mantenimiento de pequeñas cantidades de información.

Cada topics es fuertemente tipado por el tipo de mensaje utilizado por los nodos. El Master no hace cumplir la consistencia entre los mensajes de los nodos editores, pero los suscriptores no establecerán el transporte de mensajes a menos que haya concordancia en los mensajes.

ROS actualmente soporta TCP/IP y UDP para el transporte de mensajes. El transporte TCP/IP en ROS se conoce como TCPROS. TCPROS es el transporte predeterminado que se utiliza en ROS y es el único transporte en el que las bibliotecas no necesitan dependencias de ninguna clase. El transporte basado en UDP, se conoce como UDPROS y en la actualidad sólo se admiten en *roscpp*, este tipo de protocolo de mensajes es más adecuado teleoperación[9], por su bajo tiempo de latencia.

Los nodos negocian el transporte deseado en tiempo de ejecución. Por ejemplo, si un nodo de transporte prefiere UDPROS, pero el otro nodo no lo soporta, puede haber un retroceso en la configuración del transporte y cambiar de protocolo. Este modelo de negociación permite a los medios de transporte nuevos agregarse a la comunicación sin ningún problema.

10.3 Casos de Uso de ROS

En los siguientes aparatados de este capítulo se verá las posibles situaciones que se pueden darse cuando se programa tareas para robots, y como ROS a través de sus mecanismos es capaz de dar soluciones eficaces.

10.3.1 Registro y Reproducción

Cualquier flujo de mensajes de ROS puede ser almacenado en el disco y reproducirse más tarde. Todo esto puede hacerse en la línea de comandos y no requiere la modificación del código fuente de ninguna de las piezas software. Para facilitar el registro y seguimiento de los sistemas distribuidos a través de muchas máquinas, la librería rosconsole se basa en el sistema log4cxx del proyecto Apache y proporciona una interfaz en la cual todos los registros se publican en el topics rosout.

10.3.2 Visualización y Monitorización

Durante la etapa de diseño y depuración, es necesario observar los distintos estados en lo que se encuentra el programa, para ello se dispone de la herramienta de ROS Rviz. Esta herramienta es muy útil para las personas que no están familiarizados con la robótica, con ella pueden observar el proceso de creación de mapas y Waypoints. Rviz utiliza el sistema de *tf* para transformar los datos del sistema de coordenadas que llega en un marco de referencia global. Esta *tf* se verá en el apartado de transformaciones. En la *Ilustración 10.3-1*, se ve la interfaz de rviz en un proceso de Navegación en un entorno.

[9] Teleoperación. Acción de gobernar un robot ubicado en una zona remota.

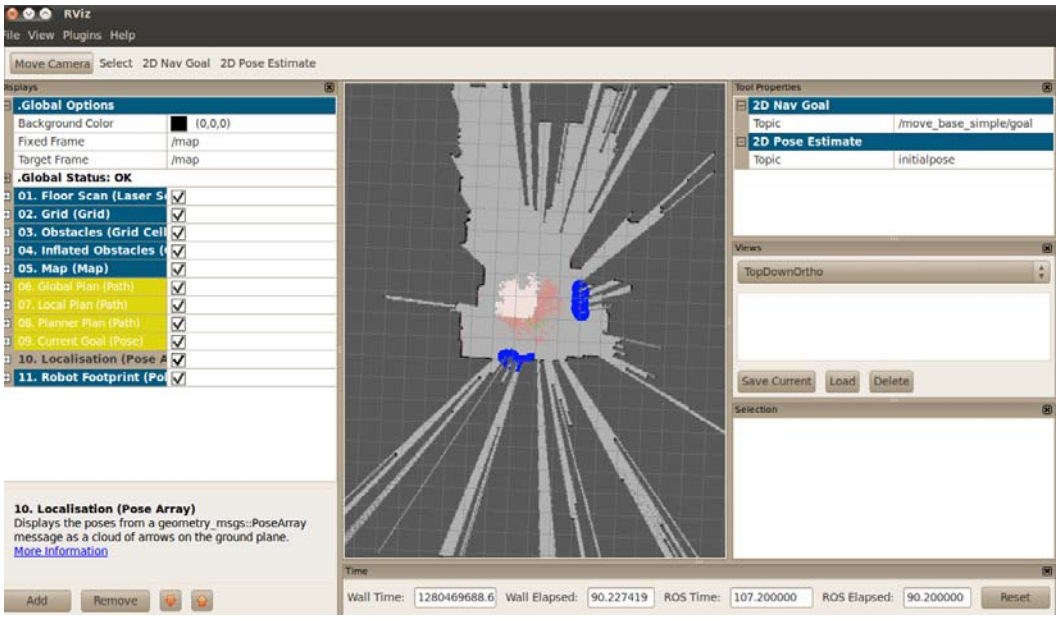

Ilustración 10.3-1. Entrono de Visualización y Monitorización RVIZ.

Existe otra herramienta para observar el cómputo de gráfico de nodos de ROS, esta herramienta se llama *rxgrahp,* que muestra los nodos que se están ejecutando así como los topics por los que se conectan. En la *Ilustración 10.3-2*, se muestra el gráfico de nodos para una aplicación que implementa el problema SLAM.

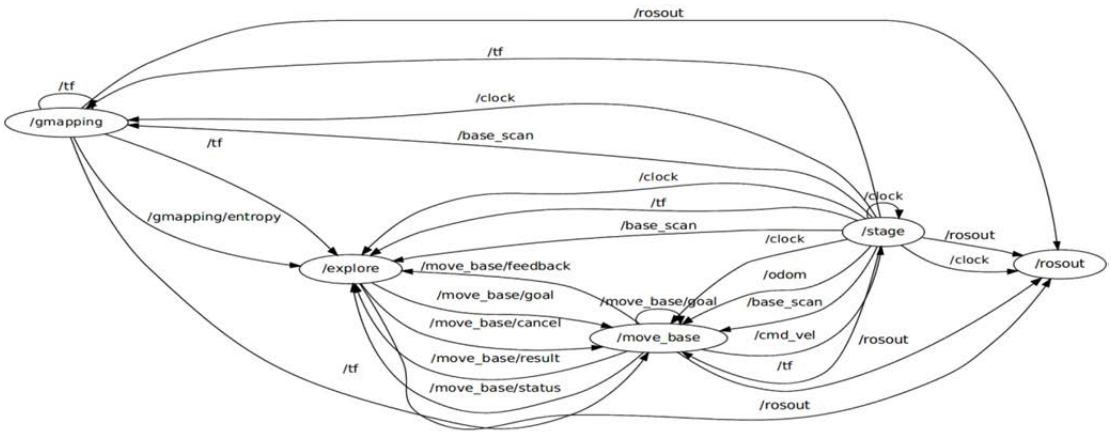

Ilustración 10.3-2. Diagrama de Rxgrahp.

10.3.3 Organización de la Información

La construcción de un sistema de control requiere de muchos procesos y funcionalidades, por eso es importante que haya una estrecha colaboración entre investigadores. Para ello ROS dispone de directorios de paquetes en los que se guarda código de algunas funcionalidades. Este código ha sido implementado por investigadores y está a disposición de todo aquel que quiera utilizarlo. Estos paquetes se organizan en directorios de la siguiente manera:

- **Package**: Es la unidad principal en la que se organiza el software en ROS. Un paquete puede contener procesos ejecutables (nodos), librerías dependientes de ROS, archivos de configuración etc.

- **Manifest**: Proporcionan información sobre un paquete, incluyendo tipo de licencia de uso y publicación, dependencias de otros paquetes e información específica de compilación.

- **Stack**: Es el conjunto de paquete que proporcionan una funcionalidad concreta, es la manera en la que el software es publicado contando con números de versión asociados. Al igual que los paquetes llevan asociado un manifiesto.

Al observar el sistema de archivos, es fácil diferenciar los Packages y Stacks, las diferencias son las siguientesa:

- Un Package es un directorio con un archivo manifest.xml.

- Un Stack es un directorio con un archivo stack.xml.

10.3.4 Grabación de Datos

Para la grabación de los datos se utilizará la herramienta de ROS Bag, esta herramienta permite guardar algunos datos de proceso y poder ser utilizados después por algoritmos, como por ejemplo algoritmos de visualización.

Los Bag los crea la herramienta ROSBAG, que suscribe a uno más topics, y almacena los datos de los mensajes serializados en un archivo .bag. Estos archivos de bag también se pueden reproducir en ROS a los mismos topics e incluso reasignarlos a nuevos topics. Aunque puede que tenga problemas con los datos almacenados en el interior de estos, debido a una mala sincronización en el tiempo de envió de mensajes. Por esta razón, la herramienta rosbag incluye una opción para publicar un reloj simulado que corresponde al tiempo de los datos que se registran en el archivo.

El formato de archivo bag es muy eficiente, tanto para la grabación y reproducción, ya que los mensajes se almacenan en la mismo formato que se utiliza en la capa de red de transportes de ROS.

10.3.5 Transformaciones

Los sistemas robóticos necesitan de la existencia de relaciones espaciales, por ejemplo: entre un robot móvil y un marco de referencia fijo para la localización, o entre varios marcos de diferentes sensores, etc. Para ello ROS dispone de la herramienta TF. Es un paquete que permite al usuario realizar un seguimiento de los marcos de coordenadas múltiples a través del tiempo. TF mantiene la relación entre los marcos de referencia en una estructura de árbol tamponada en el tiempo, y permite al usuario transformar puntos y vectores entre dos marcos de coordenadas en cualquier punto deseado en el tiempo.

TF puede operar en un sistema distribuido. Esto significa que toda la información sobre los marcos de referencia de un robot está disponible para todos los componentes de ROS en cualquier equipo del sistema. No existe un servidor central de información de transformación. La *Ilustración 10.3-3* muestra la estructura de marcos de referencia de un chasis con cuatro ruedas.

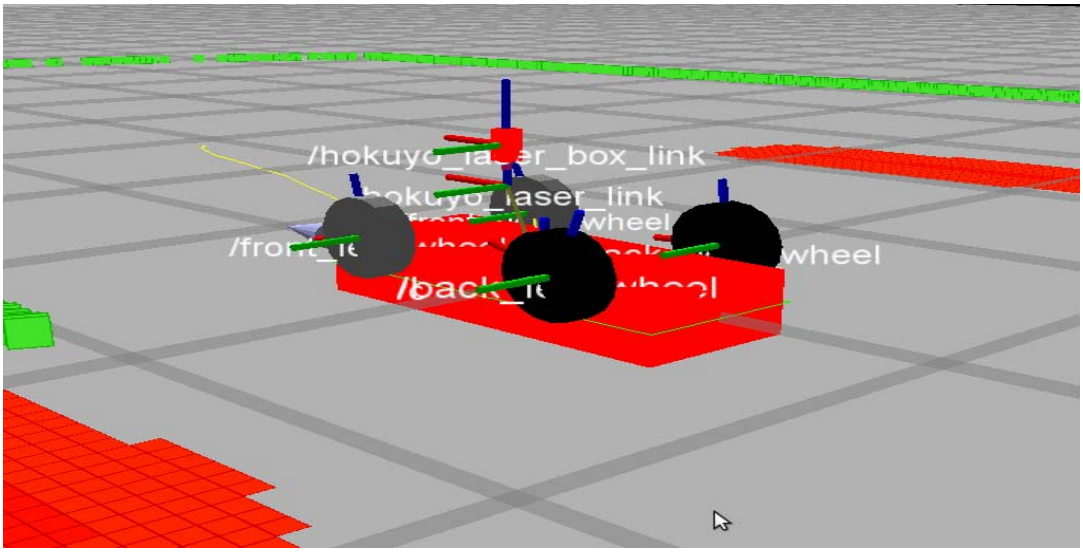

Ilustración 10.3-3. Transformaciones TF.

ANEXO IV. Hardware del Sistema

En este anexo se realizará una descripción del hardware utilizado para la realización de este proyecto.

11.1 Pionner-3DX

El robot Pionner-3DX es una plataforma popular para educación, investigación, creación de prototipos, exhibiciones y otros proyectos. Este robot puede estar equipado con un ordenador integrado en una única placa de formato EBX basada en Pentium, la cual se utiliza para comunicaciones de alto nivel y funciones de control (26).

El robot está basado en una serie de aplicaciones de cliente-servidor que contiene una serie de bibliotecas para el desarrollo de aplicaciones inteligentes. Este robot es un producto de MobileRobots. Al igual que todos los productos de esta compañía dispone de un software para el control de sus aplicaciones. Este software se llama ARIA (Interfaz de aplicaciones Robóticas), el cual está desarrollado en C++ que proporciona soporte para comunicaciones TCP/IP con el robot. La *Tabla 11.1-1* muestra las características físicas de este robot.

Pionner 3DX	
Dimension externa	450mm x 400mm x 245mm
Masa	9Kg
Máxima Carga	17 kg
Máxima Velocidad	5.76 km/h
Máximos Grados de Pendiente	25 grados
Rango de Temperatura	0 a 35 C

Tabla 11.1-1.Características Físicas del Pionner-3DX.

En la *Ilustración 11.1-1* se muestra las dimensiones del robot P3DX, estas dimensiones viene dadas en mm.

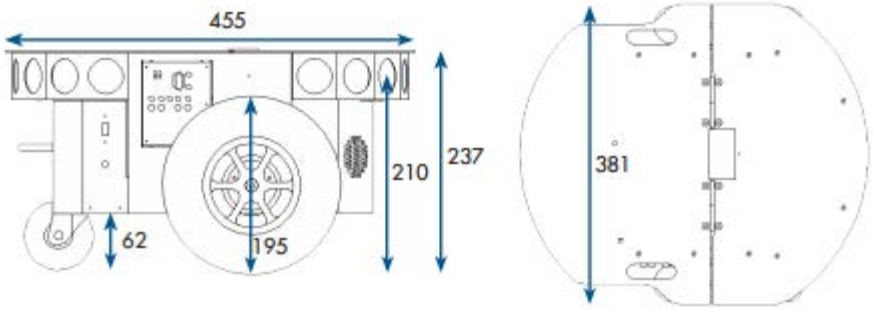

Ilustración 11.1-1. Medidas de P3DX.

Al igual que todos los robots para poder realizar sus funciones, su arquitectura se divide en distintos módulos que interactúan entre sí, como se muestra en la *Ilustración 11.1-2*

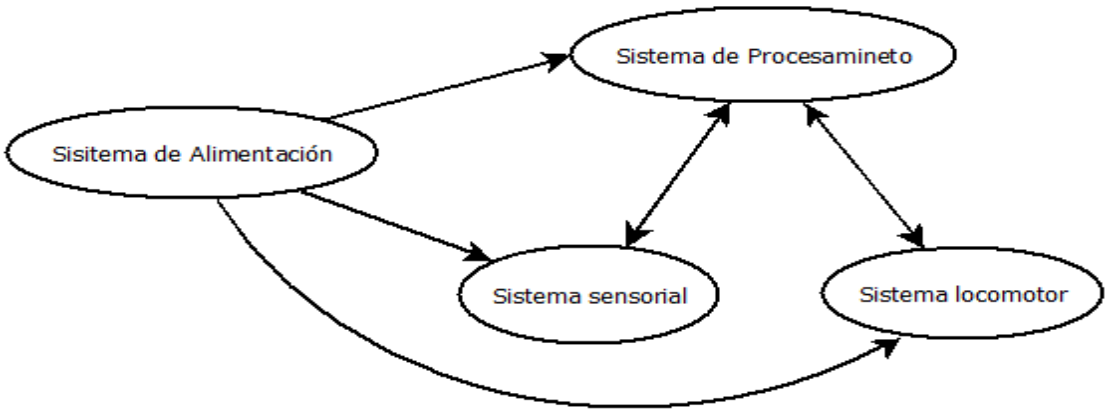

Ilustración 11.1-2. Diagrama de Iteración de Sistemas Del robot P3DX.

En los siguientes apartados se hará una descripción de los sistemas que compone la arquitectura del P3DX.

11.1.1 Sistema Sensorial

El sistema sensorial de este robot está compuesto por una serie de Bumpers y Sonar que le proporcionan información sobre las características del entorno de trabajo. El sistema sensorial del P3DX está compuesto por los siguientes elementos.

11.1.1.1 Bumpers

En el Pionner-3DX los Bumpers se dividen en dos segmentos, uno se encuentra su parte delantera y otro en la trasera. Estos Bumpers, están divididos en cinco segmentos cada uno y distribuidos en diferentes ángulos alrededor del robot. La distribución de estos segmentos es la -52, -19, 0, 19, y 52 grados alrededor del robot. A cada uno de los segmentos del Bumpers, se les asigna un nombre como muestra la *Ilustración 11.1-3*. La *Ilustración 11.1-3* muestra la disposición de los Bumpers en el robot.

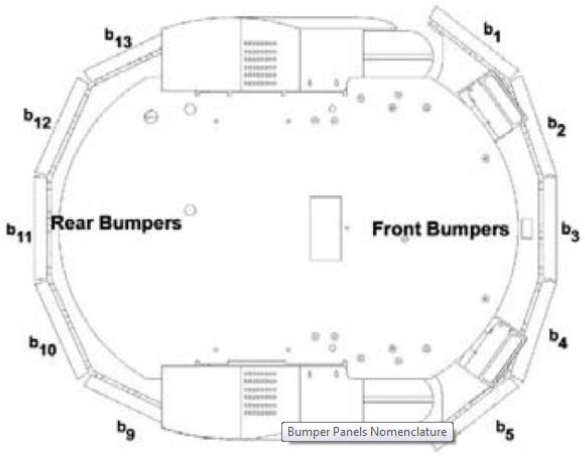

Ilustración 11.1-3. Diagramas de Distribución de Bumpers.

En la *Ilustración 11.1-4* se muestra las características técnicas de los Bumpers, en este caso será el Bumper de la parte delantera del Pionner-3DX.

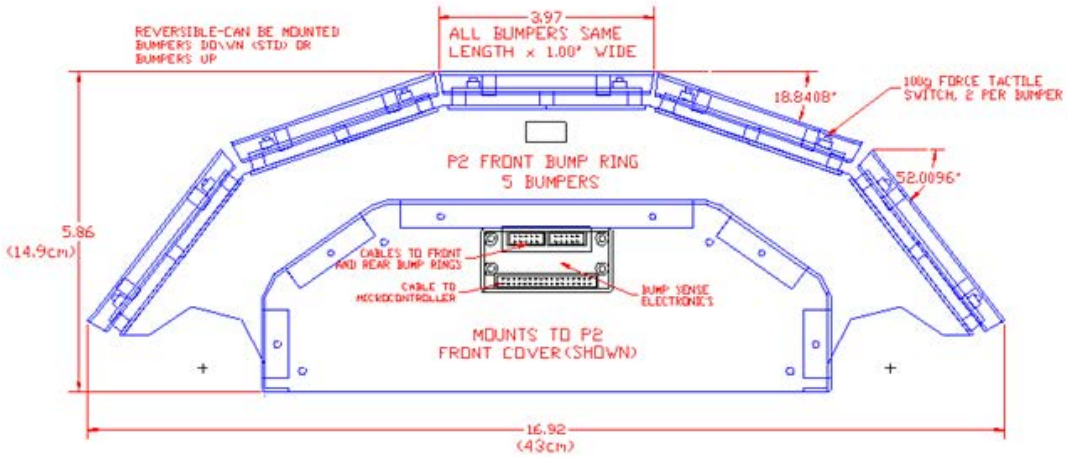

Ilustración 11.1-4. Diagrama de características de elementos de los bumpers.

La funcionalidad de los bumpers, es la de indicar si el robot colisiona con algún objeto del entorno.

11.1.1.2 Sonars

Uno de los sensores de mayor uso del Pionner-3DX es el sonar. El sonar emite una señal ultrasónica, la cual es captada de nuevo cuando choca con algún objeto. Esta señal proporciona la distancia a la que se encuentran los objetos, multiplicando la velocidad de la onda por el tiempo que tarda en recorrer la distancia. Esto proporciona una gran ventaja, debido a su reducido costo de cómputo y una elevada velocidad de respuesta. El sonar de este robot está compuesto por ocho emisores-receptores de sonidos dispuestos en la parte frontal del robot. La

información es devuelta al sistema de control en forma de array[10] de ocho posiciones. En la *Ilustración 11.1-5* se mostrará las características técnicas del sonar del robot P3DX.

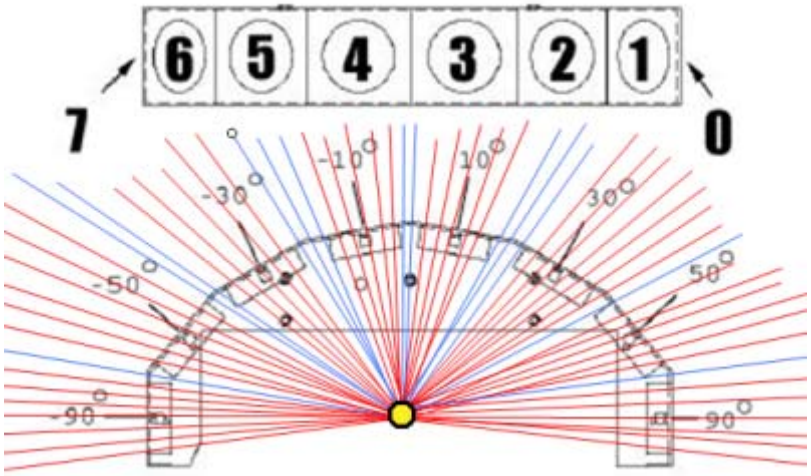

Ilustración 11.1-5. Distribución de los Sonars.

11.1.1.3 Sistema Locomotor

El sistema locomotor del robot, es el que le permite desplazarse por el entorno de trabajo. Este sistema se podría clasificar dentro actuadores y receptores. En actuadores porque reaccionan frente a órdenes del sistema de control y receptores porque da información sobre la distancia recorrida por el robot. El sistema locomotor del P3DX está compuesto por tres ruedas. Dos delanteras que sirven para la propulsión del robot y una trasera que se utilizará para el giro del robot. Los giros y desplazamientos serán madados al sistema Work-Station, para calcular las distancias y giros realizados en el entorno, y facilitar la posición del robot. Este sistema representa al sistema odométrico del robot. También está compuesto por un motor que propulsará las ruedas motrices. El motor es de la marca *Pittman brand LO-COG modelo GM8XX2 Winding Data* sus características quedan plasmadas en la *Tabla 11.1-2*.

Parámetros	Símbolos	Unidades	Valor
Voltage	V	Voltios	12.0
Torque constante	K_T	Oz · in /A	1.88
Resistencia	R_T	Ω	2.17
Inductancia	L	mH	1.57

Tabla 11.1-2. Sistema Locomotor.

11.1.1.4 Sistema de Alimentación.

[10] Array. Vector que representa un conjunto de variables del mismo tipo al cual se accede mediante índices.

Este sistema es el encargado de proveer de energía los demás sistemas de la arquitectura del P3DX, no tiene ninguna función con respecto a los actuadores y receptores salvo la dicha. El sistema de alimentación de P3DX contiene las siguientes características:

- Tiempo de duración: 2-4 horas (sin accesorios).
- Tiempo de carga: 12 horas (estándar) o 2.4 horas (opcional de alta capacidad de carga).
- Soporta hasta baterías 3 a la vez.
- Voltaje: 12 V (cada batería).
- Capacidad: 7.2 Ah (cada una).
- Química: ácido de plomo.
- Baterías intercambiables en caliente.

11.2 Microsoft Kinect

La cámara Kinect será utilizada en el proyecto para captar la información del entorno de trabajo y así poder hacer una recreación del mundo en el que se encuentra. Esta recreación se hará en forma de mapa.

El sistema de la Kinect tiene una cámara RGB, sensor de profundidad y un micrófono multi-array bidireccional que conjuntamente capturan el movimiento de los cuerpos en 3D. El sensor de Kinect reproduce video a una frecuencia de 30 Hz, en colores RGB 32-bit y resolución VGA de 640×480 pixels, el canal de video monocromo es de 16-bit, resolución QVGA de 320×240 pixels con hasta 65,536 niveles de sensibilidad. El límite del rango visual del sensor de Kinect está entre 1.2 y 3.5 metros de distancia, con un ángulo de vista de 57° horizontalmente y un ángulo de 43° verticalmente, mientras que el pivote puede orientarse hacia arriba o abajo ampliando hasta 27° (27). La *Ilustración 11.2-1* muestra la cámara Microsoft Kinect.

Ilustración 11.2-1. Cámara Microsoft Kinect.

La cámara transmite luz invisible, cercana en el espectro a los infrarrojos y puede conocer el tiempo que tarda la luz en volver al sensor tras reflejarse en los objetos. El sensor actúa como un sonar, la operación no es teóricamente complicada, si se conoce el tiempo de cada salida y llegada de la luz tras reflejarse en un objeto, sabiendo la velocidad absoluta de la luz, se puede tener la distancia a la cual se encuentra ese objeto.

En un amplio campo visual con objetos, la cámara Kinect trata de reconocer a qué distancia están los objetos, distinguiendo el movimiento en tiempo real. Kinect puede llegar a distinguir la profundidad de cada objeto con diferencias de 1 centímetro y su altura y anchura con diferencias de 3 milímetros. El hardware de Kinect está compuesto por la cámara y el proyector de luz infrarroja, añadido al firmware y a un procesador que utiliza algoritmos para procesar las imágenes tridimensionales.

ANEXO V. Diseño Detallado de la Aplicación ANEXO IV

La implantación de este proyecto está basada en las bibliotecas que ofrece el framawork ROS. A pesar de esto hay que realizar la programación del servicio del controlador. Esto se debe a que se necesita crear funcionalidades propias que permitan alcanzar los objetivos del proyecto. También se necesita modificar los parámetros de entrada de algunos servicios para adecuarlos a las necesidades del proyecto. Este anexo contiene el diseño detallado de la aplicación, describe las respectivas clases, sus métodos y parámetros.

La descripción de estas clases se realizará por componentes de Work-Station, que será cada uno de los apartados que conforman este anexo.

12.1 Controlador

En este apartado se realiza la descripción de las clases del Controlador. El Controlador está compuesto por las siguientes clases:

- **RosAria.cpp:** Esta clase realiza dos acciones, andar y girar. Impulsará el robot hacia adelante con seguridad, mientras Turn evita los obstáculos detectados por el sonar. Esta clase implementa acciones como puede ser, girar moverse, etc. Cada una de estas acciones tienen un constructor y un destructor. Cada acción, también implementa un método virtual [11]esencial, este método se llama fire(). Los métodos que contiene esta clase son los siguientes:

  - **enableMotors():** Realiza el encendido de motores del robot siempre y cuando la batería tenga la energía suficiente**.**

  - **Device():** Se encarga de establecer los parámetros para los distintos dispositivos del robot.

  Parámetros:

  - **Id_port**: Es un string[12] y representa al puerto por el que se hará la conexión entre el robot y Work-Station, en este caso tiene como valor por defecto "/dev/ttyUSB0"

[11] Método virtual. Este método permite cambiar definiciones de una clase padre en una clase hija.
[12] String. Se denomina en programación a una cadena de caracteres.

- **Velociadad_maxima:** Representa la velocidad máxima que alcanzará el robot cuando no tenga ningún obstáculo cerca.

- **Distacia_frenado:** Este parámetro representa la distancia a la que se frenará el robot cuando detecte un obstáculo.

Los objetos ROS instanciados en esta clase son:

- **nav_msgs**: Esta objeto sirve para posicionar al robot en el espacio. Pasará al sistema de ROS los metros en las direcciones en las que se desplaza el robot.

- **geometry_msgs**: Obtiene la velocidad y el giro que realiza el robot para poder así situarlo en el mapa.

- **ActionGo:** Esta clase tiene como principal funcionalidad dar la velocidad máxima que puede alcanzar el robot y a la distancia mínima a la que debe parase ante un obstáculo localizado por el sonar. Esta clase obtiene el array que devuelve el sonar con las distancias. Si una de las distancias es igual o menor a la distancia mínima permitida, se parara y ejecutará la acción de girar. Los parámetros que se pasan a esta clase son la velocidad máxima y la distancia a la que debe modificar su trayectoria cuando encuentre un obstáculo.

- **ActionTurn:** Esta clase define la actuación del robot cuando encuentra un obstáculo. Se le pasa como argumentos la cantidad a gira y la velocidad. Al igual que la clase anterior, lee datos del sonar y obtiene la posición en la que se encuentra el obstáculo, así puede girar hacia el lado contrario en el que se encuentra el obstáculo.

- **ActionRecover:** Esta clase implementa la funcionalidad de retroceder y girar cuando los bumpers del robot colisionan con algún obstáculo que no ha sido detectado por el sonar. Se le pasa como parámetros la distancia a retroceder en caso de choque.

- **ActionPause:** Esta clase gestiona las paradas del robot en el caso de cómputo y señal de colisión de los bumpers. Actúa principalmente como semáforo que activa y desactiva la señal de movimiento de las ruedas del robot**.**

- **P2os_driver:** P2os_driver proporciona un controlador para robots utilizando el interfaz de P3DX P2OS/ARCOS en la forma de un nodo de ROS. Se utilizará esta clase para obtener información sobre los dispositivos del robot. Las funciones que componen esta clase son las siguientes:

  - **Subscribe():** Este método se utilizan para controlar las suscripciones al driver, el driver podrá anular esta subscripciones, pero por lo general no lo hará. Este método recibe como parámetro el *id* del driver.

  - **Setup():** Inicializa las subscripciones a los dispositivos. Este método es llamado por subcriber() en caso de que no se produzca ningún error este método devolverá 0.

- **Shutdown():** Este método es el encargado de finalizar el driver y las subscripciones a los dispositivos.

Está en clase, se hace referencia en la clase Move_base cuando crea un plan**.** Como se puede apreciar en la *Ilustración 12.4-1* del apartado Navegación de este capítulo**,** corresponde con el módulo base controller, que se explicará más adelante. P2os_driver recibe de Move_base la velocidad a la que debe moverse y la dirección que debe seguir el robot.

- Esta clase se llama **Keyboard.cpp** y tiene como objetivo desplazar al robot, basándose en un teclado de ordenador, para ello, se le deberá asignar a cada tecla una funcionalidad. La relación tecla funcionalidad es la siguiente:

  - **"WS":** Estas teclas servirán para desplazar al robot hacia delante y hacia atrás respectivamente.

  - **"AD":** Mueve a izquierda y a derecha el robot, la tecla A hacia la izquierda y D hacia la derecha.

  - **"04":** El 0 es para parar el motor y el 4 para encenderlo, cabe destacar que si el motor no está encendido, el robot no se moverá en ninguna dirección aunque se pulse alguna tecla.

  - **"Shift":** Si se pulsa esta tecla y W o S conjuntamente el robot se desplazará a máxima velocidad, que será 0.5 m/s.

  Además, esta clase instanciará un objeto de las clases nav_msgs y geometry_msgs, que son las mismas que instancia RosAria.cpp.

12.2 Visor del Entorno

En este apartado se realizará la descripción de la clase que compone este módulo. La clase es la siguiente:

- **Openni_kinect**: Esta clase es la encargada del control de los datos de la cámara Microsoft Kinect. Es establece la conexión con el hardware y obtienen los datos del entorno. También publica varios Topics para que otras clases puedan acceder a la información e interactuar con ella. Este es el contenido de la clase Openni_kinect:

  - **Topic:**
    - `rgb/camera info`: Este método sirve para la calibración e información de metadatos.
    - `rgb/image raw`: Obtiene la imagen del dispositivo. El formato es Bayer GRBG para Kinect.

- **Servicios:**
  - `rgb/set_camera_info`: Ajusta la calibración de la cámara RGB.
  - `ir/set_camera_info`: Ajusta de calibración de la cámara IR.

- **Parámetros**:
  - `device id`: Especifica qué puerto debe abrirse. Los formatos disponibles son:
    - # 1: Usar el primer dispositivo encontrado.
    - @ 3: Utiliza el puerto USB bus 2, dirección 3.
    - B00367707227042B: puerto con número de serie dado.
  - `time_out`: Comprueba cada segundo del `~time_out` si las cámaras activas han transmitido nuevos datos.
  - `rgb_frame_id` (default: `/openni_rgb_optical_frame`): El marco referencia de `tf` de la cámara RGB.

12.3 Creación de Mapas

El módulo de Creación de Mapas está compuesto por las siguientes clases:

- **Slam_gmaping:** Esta clase sirve para crear un mapa en 2D representa al componente Gmapping de la arquitectura descrita en el capítulo Diseño de la solución Técnica.

  - **Topic:**
    - `map_metadata`: Actualizará los datos del mapa a construir.
    - `Entropy`: Estimación de la entropía de la distribución de la pose del robot (un valor más alto indica mayor incertidumbre).
  - **Servicios:**
    - `dynamic_map`: Servicio de la obtención de los datos de los mapas.
  - **Parámetros:**
    - `base_frame`: Enlace al bastidor de la base móvil de robot.
    - `odom_frame`: Enlace al sistema odométrico.
    - `temporalUpdate`: Procesar un análisis, si la última exploración es más antigua que el tiempo de actualización en segundos, indicado en este parámetro.
    - `maxRange`: El rango máximo del sensor. Marca el final de regiones libres a reconocer por el sensor.

- **Map_server:** Esta clase se encarga de crear la imagen del mapa del entorno. Esta clase representará una imagen en mapa de bits en formato .png, cada pixel representa una celda y el color de cada pixel el estado. Existen tres estados ocupado, libre o desconocido. En el caso de las imágenes en escala de grises, el umbral se hace respecto al valor de intensidad de cada pixel. En el caso de las imágenes a color cada pixel se le compara en base a su valor de ocupación definido:

- $$occ = \frac{255 - color\_avg}{255}$$

12.3-1

Donde *color_avg* corresponde al promedio de intensidad de todos los canales de la imagen. Una vez terminada la exploración, esta clase guardará la imagen del mapa y el fichero que contiene información como la resolución y el tiempo de creación del mapa. La ruta del directorio en el que se almacena esta información será pasada por parámetro.

- **Waypoints:** Esta clase lo que hace es establecer los waypoints en los lugares libres de obstáculos, para ello coge la información del mapa proporcionado por la clase Map_server, divide este mapa en celdas que pueden ser proporcionadas por el usuario o toma un valor por defecto. Si la celda está vacía, establece un Waypoint en el centro de esta. Esta clase está compuesta por los siguientes parámetros y métodos:

  - Parámetros:

    - `width`: Número de pixeles del ancho del mapa.
    - `height`: Número de pixeles del alto del mapa.
    - `total`: Número de pixeles totales del mapa.
    - `contadorllamadas`: Número de llamadas realizadas al servicio que proporciona el mapa, sirve para que el proceso no repita operaciones y sea más eficiente.
    - `metrosYreconocidos`: Metros del eje Y reconocidos y clasificados como ocupado o libre.
    - `metrosXreconocidos`: Metros del eje X reconocidos y clasificados como ocupado o libre.
    - `metrosYtotal`: Metros del eje Y totales.
    - `metrosXtotal`: Metros del eje X totales.
    - `casillaLibre`: Muestra si la celda de la rejilla está libre de obstáculos
    - `ladopiselPixel`: La resolución son m/pixel, los metros cuadrados que representa un pixel.
    - `centroPixelXY`: Posición del centro del pixel, como es cuadrado basta con representarlo con un double.
    - `p_entera`: Parte entera de la división del lado de una celda de la rejilla.
    - `const char * rutaFicheroXML`: Ruta del fichero .XML donde se guardará la información de los Waypoints.
    - `tamanyoX_explora`: Metros del eje X que el usuario quiere explorar. Si el usuario no introduce nada, será un valor por defecto de 50 metros.
    - `tamanyoY_explora`: Metros del eje X que el usuario quiere explorar. Si el usuario no introduce nada, será un valor por defecto de 50 metros.
    - `metros2reconocidos`: Metros cuadrados reconocidos.
    - `mapacompletado`: Este booleano marca si el mapa ha sido explorado por completo.
    - `std::list<Waypoints > waypoints`: Lista que almacena las posiciones de los waypoints y las distancias con los demás Waypoints.

- Métodos:

  - `void divisiones()`: Realiza la división del mapa en celdas. El tamaño de las celdas es fijado por el usuario, se le da un valor por defecto en caso de que no sea así.
  - `void obtencionCentroCelda()`: Obtiene el centro de la celda de la rejilla fijada en el anterior mapa. Estos dos métodos se ejecutan una vez, por eso se ha establecido el parámetro controllamadas.
  - `void transformaMapa()`: Este método lo que hace es crear los waypoints del mapa, para ello recorre el mapa pixel por pixel estableciendo si están ocupados, libres o son desconocidos, a la vez comprueba si ha realizado el análisis de una celda fijada por la rejilla creada en divisiones. Si ha realizado el análisis de una celda y esta tiene todo sus pixeles vacíos establece un waypoint en el centro de la celda. Cabe destacar que el mapa se recorre de arriba abajo, por lo cual, habrá que cambiar la posición de los píxeles para que corresponda con las medidas, para ello se realiza la siguiente operación:

    ```
    unsigned int i = x + (alto - y - 1) * ancho;
    ```

  - `void Callback(const nav_msgs::OccupancyGrid::ConstPtr& map)`: Llama al servicio de ROS que proporciona la información del mapa creado por la clase Map_server.
  - `void buildDomainXml()`: Este método es el encardo de realizar el fichero xml con la información del mapa, la posición del robot y los waypoints y sus conexiones.
  - `void estableceConexiones()`: Realiza la conexión de los waypoints por medio de la distancia euclídea entre estos. La fórmula de la distancia euclídea para un espacio bidimensional de la distancia euclidiana entre dos waypoints W_1 y W_2, de coordenadas (x_1, y_1) y (x_2, y_2) respectivamente, es:

$$d_E(W_1, W_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad 12.3\text{-}2$$

Cabe destacar que esta clase establece las conexiones con respecto a las casillas de la rejilla adyacentes a las que pertenece W_1.

  - `bol trataArgumentos(ros::NodeHandle)`: Este método trata los argumentos pasados por el usuario. En caso de que sean incorrectos manda un mensaje de error y establece unos parámetros por defecto si estos son nulos.

- **Xml_waypoints:** Está clase se encargará de crear un mensaje para que cualquier sistema que esté basado en ROS, este mensaje tendrá la siguiente información:

  - `terminado_mapeo`: Este parámetro es un booleano, representa si la superficie a mapear ha sido completada;
  - `time_publisher`: Tiempo según el reloj de ROS en el que ha sido mandado el mensaje.

- `mensaje_pub`: Este mensaje contiene el XML creado en la clase Waypoints.

12.4 Navegación

En ese apartado se describe el módulo de Navegación, las clases de este módulo son las siguientes:

- **Move_base**: Esta clase es la encargada de mover al robot por un mapa previamente creado. La ejecución del nodo Move_base en un robot que esté correctamente configurado hará que éste intente alcanzar una meta con una tolerancia especificada por el usuario. En el caso de que se encuentre con obstáculos que no estaban en el mapa, realizará una serie de acciones. La primera acción será añadir esos obstáculos al mapa ya creado anteriormente. También modificará su trayectoria para evitar el obstáculo y poder alcanzar la meta marcada. Si se observa que aparecen muchos obstáculos donde el mapa no ha indicado nada, entonces mandara un mensaje de error, ya que no reconoce el mapa con el entorno en el que está. Esta clase se ayudará de AMCL, que sirve para posicionar al robot en el mapa. Las partes que componen esta clase son las siguientes:

  - **Topics:**
    - `move_base_simple/goal`: Proporciona una interfaz de esta interfaz será la que se utilizará para marcar la posición del Waypoint meta.
    - `cmd_vel`: Velocidad a la que debe moverse el robot en la navegación base móvil.

  - **Services:**
    - `make_plan`: Sirve para que un usuario pueda pedir información sobre un plan sin ejecutar ese plan.
    - `clear unknown space`: Permite explorar sitios desconocidos en el área que rodea al robot. Esto es útil cuando Move_base tiene sus costmaps detenidos por un período largo de tiempo.

  - **Parámetros:**

    - `base_global_planner`: El nombre del plugin para que el planificador de cambios global pueda usar `move_base`. Este plugin debe adherirse a la interfaz de `nav_core :: BaseGlobalPlanner` se especifica en el empaque `nav_core`
    - `base local planner`: Sirve para que se pueda utilizar el planificador de cambios local. Al igual que el otro parámetro, este plugin debe adherirse a la interfaz de `nav_core::BaseGlobalPlanner` se especifica en el empaque `nav_core.recovery_behaviors:`
    - `controller frequency`: El número en Hz en el que ejecuta el bucle de control y se envían comandos de velocidad a la base.
    - `shutdown_costmaps`: Determina, si o no, a la parada de los costmaps del nodo, cuando `move_base` está en un estado inactivo

- `oscillation_time`: Tiempo en segundos que permitir la oscilación antes de ejecutar las conductas de recuperación. Un valor de 0,0 corresponde a un tiempo de espera infinito.
- `outoscillation distance`: Distancia en metros que el robot debe moverse teniendo en cuenta que no debe oscilar. Si se avanza en este momento se restablece el temporizador de cuenta, hasta el `oscillation_timeout`.
- `planner_frequency`: El número de Hz en el que se ejecutará el bucle de planificación global. Si la frecuencia está ajustada a 0,0, el planificador global sólo se ejecutará cuando se reciba del planificador local la notificación de que hay un objeto en su trayectoria.

En la *Ilustración 12.4-1* muestra el funcionamiento de esta clase y la interacción entre módulos de ROS que se realiza (19).

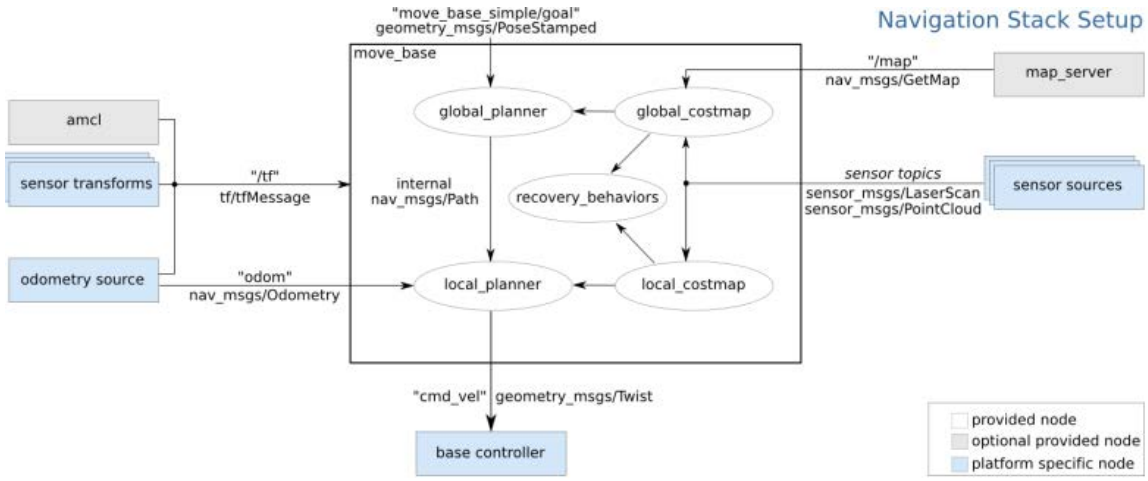

Ilustración 12.4-1. Funcionamiento de la clase Move_base.

Este diagrama muestra la arquitectura entre módulos que interactúan con la clase Move_base. Este diagrama consta de tres tipos de módulos; los módulos de Entrada, los módulos de Salida y los módulos internos, que son los que componen la clase Move_Base. A continuación, se describen cada uno de ellos:

- **Entrada**: Estos módulos proporcionan la información necesaria para crear un plan. Esta información es la creada en las clases Amcl y Map_server, así como la información obtenida del sistema odométrico y los distintos dispositivos, que en este caso sería el láser de la Kinect.

- **Salida**: Estos módulos proporcionan la salida, que es la velocidad y la distancia que debe recorrer el robot. Esta información de salida es pasada al módulo base_controller, que en este proyecto representa a la clase P2os_driver.

- **Internos**: Los módulos internos son aquellos que compone la clase la clase Move_base, cada uno de estos realiza una función que es la siguiente:

- Global_planner: El planificador global es responsable de la generación de un plan de alto nivel para la ruta de navegación a seguir. Dado un objetivo que está arbitrariamente lejos del robot, el planificador global creará una serie de puntos de referencia para el planificador local.
- Local_planner: Este módulo proporciona las creaciones de trayectorias y enfoques dinámicos de navegación para el robot. Dado un plan a seguir y un costmaps, el Local_planner produce comandos de velocidad para enviar a una base móvil.
- Recovery_behaviors: Restablece un nuevo plan en caso de que el robot se atasque en algún punto.
- Global_costmap: Obtiene del mapa los obstáculos lejanos al robot para poder realizar un plan global.
- Local_costmap: Obtiene los obstáculos cercanos al robot para poder realizar un plan local.

Tanto la planificación global como local de esta clase está basada en Grafos de Visibilidad descritos en el apartado 2.4.2.2.

- **Amcl**: Esta clase implementa el algoritmo de Monte Carlo descrito en el apartado 2.5.1.4. Esta clase, dado el marco de referencia /odom y el /map establece una conexión entre estos. Esta clase coge un mensaje del láser y del mapa he intenta estimar la posición en la que se encuentra el robot. En su primera ejecución cogerá los valores por defecto que son los definidos en los parámetros, estos parámetros establecerán una nube de puntos de tamaño moderado. Los topic, Publisher y Services de ROS de esta clase son los siguientes:

  - Topic:

    - `laserScan`: Este topic da la información que se recoge del láser de la Kinect.
    - `tf`: obtiene todas las transformaciones de sistema.
    - `initialpose`: Pose inicial en la que se estima que esta el robot.
    - `map`: este topic proporciona los datos del mapa que se creó anteriormente.

  - Services:

    - `global_localization:` Inicia la localización global, en el que todas las partículas son dispersadas aleatoriamente a través del espacio libre en el mapa.

  - Publisher:

    - `amcl_pose:` Publicación de la posición estimada en el mapa según esta clase.
    - `Particlecloud`: El conjunto de estimaciones planteadas por el filtro de partículas.
    - `Tf:` Transformación del /odom con la nueva estimación de la posición.

Amcl busca la transformación entre el marco del láser y el bastidor base (~ base_frame_id), y se engancha para siempre. Así AMCL puede manejar un láser que se mueve con respecto a la base.

El dibujo de abajo muestra la diferencia entre la localización utilizando odometría y AMCL. Durante el funcionamiento AMCL se estima la transformación de la estructura de la base (~ base_frame_id) con respecto al marco global (~ global_frame_id) pero sólo se publica la transformada entre el bastidor y el marco global de odometría (~ odom_frame_id) (19).

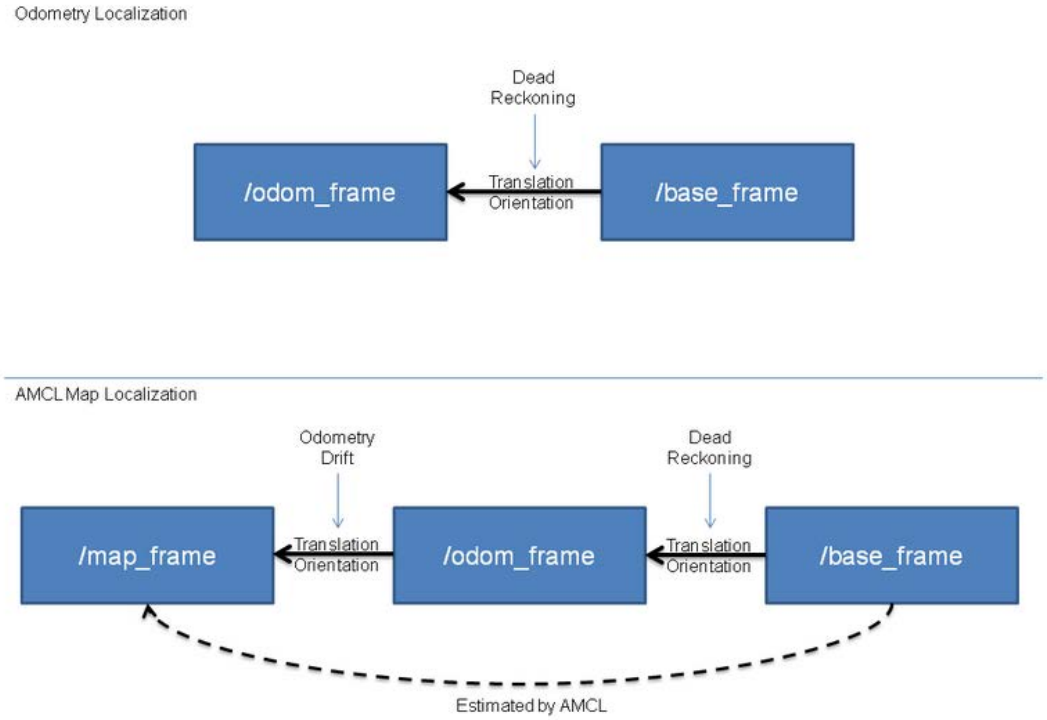

Ilustración 12.4-2. Diferencia de localización entre odometría y AMCL .

Como se aprecia en la *Ilustración 12.4-2* el marco de referencia /base_frame representa la transformada entre el robot y el infrarrojo de la Kinect, luego las posiciones tomadas por el láser serán referenciadas como las distancias de los obstáculos a la base del robot. Como se ha dicho anteriormente en el apartado 4.3, las coordenadas de referencia son tomadas como las mismas en los marcos de referencia de /base_frame y /odom_frame. Una vez realizadas las trasformaciones entre los marcos de referencia /base_frame y /odom_frame queda realizar la transformación con el marco de referencia del mapa que es /map_frame, se necesita de esta transformación para realizar los movimientos hasta el objetivo, ya que el usuario marca dicho objetivo en el mapa. En el caso de la localización, sólo es necesario realizar una transformación entre el infrarrojo y el sistema de odontometría, debido a que no hay que alcanzar un objetivo descrito en un mapa.

- **Navegation**: Esta clase se encargará de obtener del usuario el waypoint meta y buscar en el fichero XML la las referencias del waypoint. Una vez que tenga estas referencias, las publicará a través del Publisher `move_base_simple/goal` de la clase Mobe_Base. Esta clase esta clase está compuesta por los siguientes componentes:

    - Parámetros:
        - `goal_y`: Representa la posición meta del eje Y de coordenadas del mapa
        - `goal_x`: Representa la posición meta del eje X de coordenadas del mapa

        - `waypoint_meta`: El número de waypoint que el usuario ha introducido como meta.
        - `ruta_fichero`: Representa a la ruta del fichero donde están guardados todos los waypoints.
    - Métodos:
        - `get_pose`: Este método lo que haces es obtener de fichero XML la posición meta.
    - Publisher:
        - `move_base_simple/goal`: Publicará las coordenadas de (x, y) del estado meta.

12.5 Transformación

Este módulo está compuesto por la siguiente clase:

- **Pionner_tf:** Esta clase lo que realiza es una transformación y rotación mediante cuaterniones de los marcos de referencia del robot y la cámara Kinect. En el caso de este proyecto solo se realiza una translación, esto es debido a que el marco de referencia de la Kinect tiene la misma orientación que el del robot. Para obtener la información de las transformaciones es preciso mirar el aparatado 2.5 y el Anexo III. ROS.

  La siguiente figura se muestra la translación que hace esta clase entre los marcos de referencia /base_link y /base_laser que es el marco perteneciente a la cámara Kinect**.**

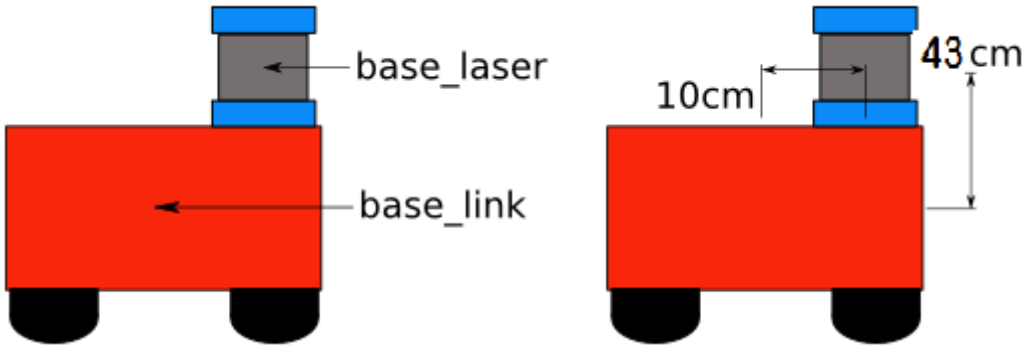

Ilustración 12.5-1. Translación Clase Pionner_tf.

Todo esto proporciona la parte de desarrollo del proyecto. Se ha seguido este desarrollo porque con él se cumplen todos los requisitos del sistema. También es la forma más eficiente de trabajar, ya que se basa en ROS y varias funcionalidades están ya implementadas.

La *Ilustración 12.5-2* muestra la iteración de las clases de este anexo, así como sus atributos y funcionalidades.

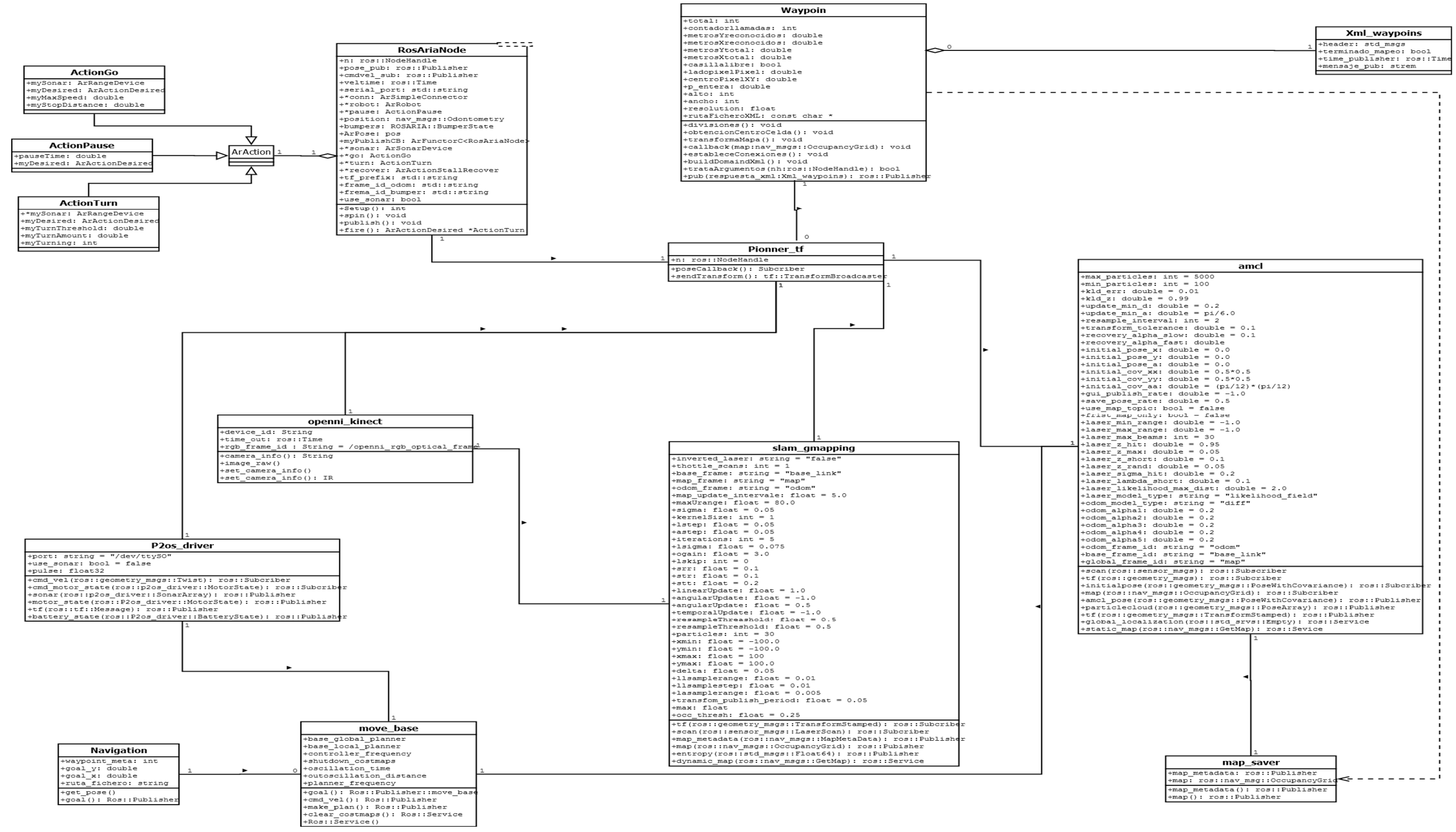

Ilustración 12.5-2. Diagrama de clases.

Anexo. Instalación de la cámara Microsoft Kinect a la batería del P3DX

En este anexo se describe los paso para instalar la cámara Microsoft Kinect a la corriente eléctrica del robot P3DX (19). Para realizar esta conexión se necesita los siguientes requisitos:

- Regulador de voltaje de 12V 1A (por ejemplo NTE966).
- Condensadores 0.33µF y 0.1µF.
- Una placa de prototipo (por ejemplo SchmartBoard 201-0001-01).
- Un conector DB25.
- Tubo termorretráctil para encubrir las soldaduras

Una vez que se dispone de estos elementos, se necesita construir un regulador de 12V. Para ello, se construirá el siguiente circuito que muestra la *Ilustración 12.5-3.*

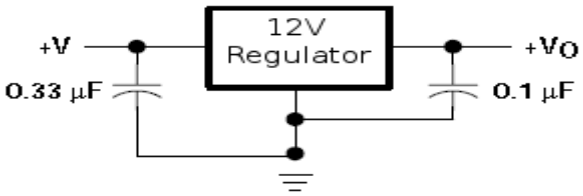

Ilustración 12.5-3. Regulador de 12V

Una vez visto esto, lo que se hace es soldar los condensadores 0.33µF y 0.1µF y el regulador NTE966 a la placa prototipo SchmartBoard 201-0001-01. Como muestra la *Ilustración 12.5-4.*

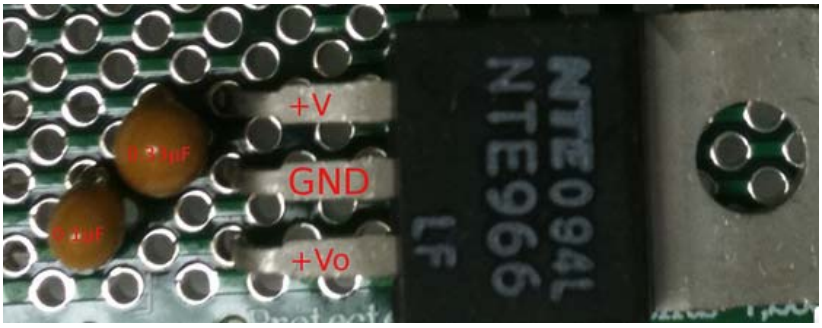

Ilustración 12.5-4. Montaje en la placa SchmartBoard

El siguiente paso es pelar el cable de corriente eléctrica de la Kinect. El cual, tendrá un cable de tierra de color blanco y otro de +12V de color marrón, como muestra la *Ilustración 12.5-5*.

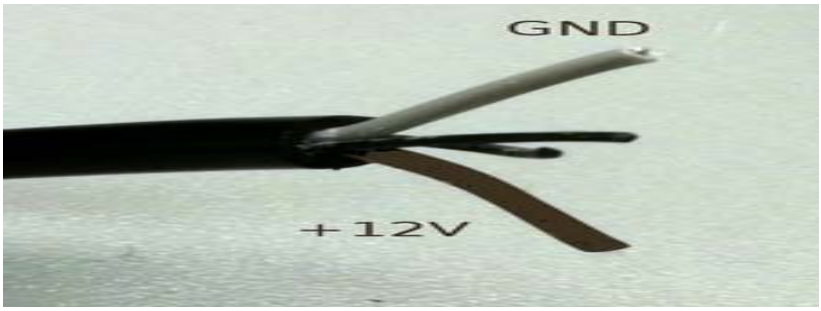

Ilustración 12.5-5. Cables Kinect

Se suelda los cables de corriente de la Kinect GND y +12V a la placa prototipo SchmartBoard 201-0001-01 en las conexiones GND y +Vo, respectivamente. Una vez que se tiene esto, se necesita dos cables uno de tierra GND y otro para la conexión +V, estos dos cables serán los que se conecten a la batería del robot. Cuando se realice esto, se recubrirá con cinta termorretráctil para evitar posibles accidentes. Véase la siguiente *Ilustración12.5-6*.

Ilustración 12.5-6. Cables soldados a la placa SchmartBoard.

Para conectar los cables a la batería del robot se necesita de un conector DB25 Cuyas características físicas a parecen en la *Ilustración 12.5-7*.

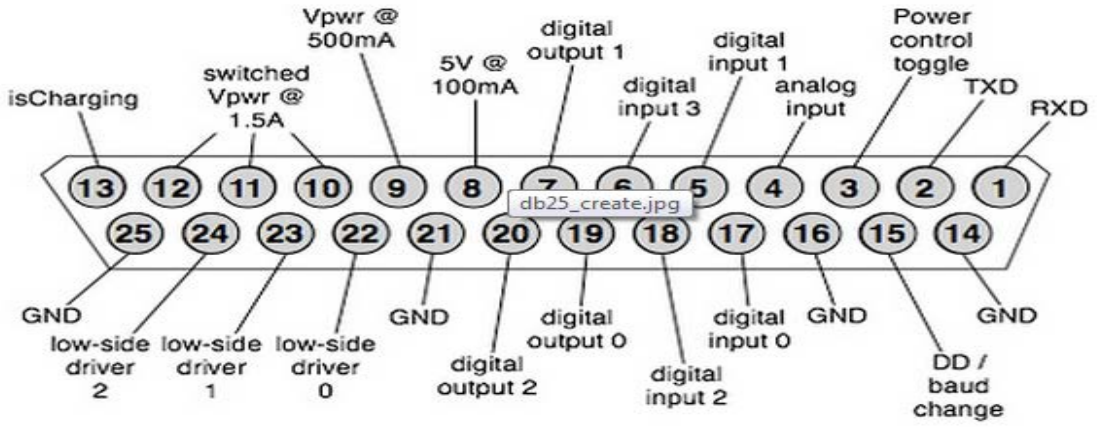

Ilustración 12.5-7. Conector DB45

Se conectará el cable GND al pin 14 y el cable +V al pin 10. Se recubre estas conexiones con tubo termorretráctil, y se engancha este conector a la batería de P3DX. Véase la *Ilustración 12.5-8.*

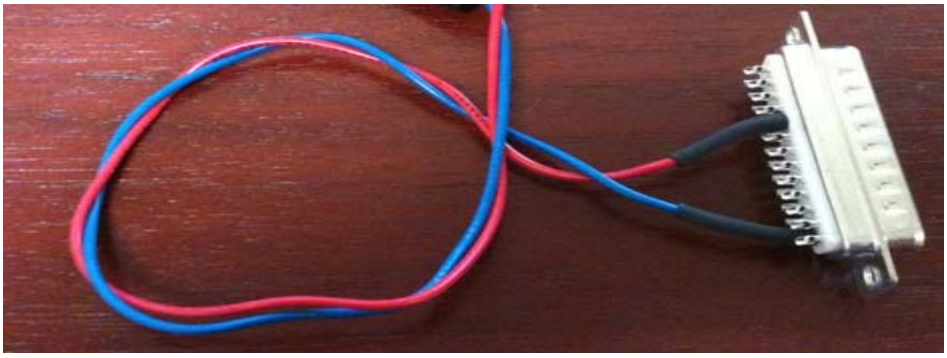

Ilustración 12.5-8. Cables al conector DB25

Acrónimos

• API: Application Programming Interface.
• ARIA: Interfaz Avanzado de Aplicaciones Robóticas.
• ASP: Active Server Pages.
• CRG: Cilindros Rectilíneos Generalizados.
• DLL: Dynamic Linking Library.
• DSS: Decentralized Software Services.
• GPS: Global Positioning System.
• GNU: Network Object Model Environment.
• GTK: GOBI Tool Kit.
• HTTP: HyperText Transfer Protocol.
• IA: Inteligencia Artificial.
• ISO: International Organization for Standardization.
• LISP: LIStas Proceso.
• LRF: Laser Range Finder.
• MSRDS: Microsoft Robotics Developer Studio.
• RFS: Radio Frequency Systems
• RGB: Red – Green – Blue.
• ROS: Robot Operation Systems.
• SO: Sistema Operativo.
• SONAR: Sound Navigation And Ranging.
• SSh: Sound Navigation And Ranging.
• TCP/IP: Transfer Control Protocol / Internet Protocol.
• UDP: User Datagram Protocolo.
• XML: Extensible Markup Language.

Bibliografía

1. *Boletín Informativo.* **Producción, Asociación de Robótica Automatización de tecnologías de la.**

2. **José A. Castellanos, Juan D. Tardós.** *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach.* s.l. : Kluwer.

3. **Patnaik, Srikanta.** *Robot Cognition and Navigation: An Experiment with Mobile Robots.* s.l. : Springer.

4. **S. Sitharama Iyengar, Alberto Elfes.** *Autonomous Mobile Robots: Control, planning, and architecture.* s.l. : IEEE Computer Society Press.

5. **Fuentes Ríos, Azucena.** Localización y modelado simultáneos en ROS para la plataforma robótica Manfred. *Proyecto Fin de Carrera.*

6. *Robot mapping: Exploring Artif cial Intelligence in the New Millenium.* **Kaufmann, S. Thrun. A survey. Morgan.** 2002.

7. *Using occupancy grids for mobile robot perception and navigation.* **Computer, A. Elfes..** 1989.

8. **Chang, Her-Jye.** *Simultaneous Localization and Mapping Algorithms with Environmental-structure Prediction for Single- and Multi-robot Systems.* s.l. : ProQues.

9. **F. Alonso Amo, Loïc Martínez.** *Introducción a la Ingeniería del Software: Modelo de Desarrollo de Programas.* s.l. : Delta Publicaciones.

10. **John Stephen Mullane, Ba-Ngu Vo,Martin David Adams,Ba-Tuong Vo.** *Random Finite Sets for Robot Mapping & SLAM: New Concepts in Autonomous Robot Representation.* s.l. : Springe
r.
11. **Likhachev, D. Ferguson and M.** *Efficiently using cost maps for planning complex maneuvers.* s.l. : Lab Paper(GRASP).

12. **López, Alejandro.** Navegación global utilizando grafo de visibilidad. *Proyecto Fin de Carrera.* 2005.

13. *Probabilistic robotics.* **S. Thrun, W. Burgard, and D. Fox.** s.l. : MIT Press.

14. *Dot pattern processing using Voronoi polygons as neighborhoods.* **Ahuja, N.** s.l. : Proceedings 5th Int. Conf. on Pattern Recognition.

15. *Estimating Uncertain Spatial Relationships* . **Randall Smith, Matthew Self, Peter Cheesemans.** California : s.n.

16. **Ming Xie, Youlun Xiong, Caihua Xiong, Zhencheng Hu.** *Intelligent Robotics and Applications* . Singapore : Springer, 2009.

17. **North, G. Welch and G. Bishop. r.** *An introduction to the kalman f lte.*

18. **Doucet, A., De Freitas, N. y Gordon, N.J.** Sequential Monte Carlo Methods in Practice. s.l. : Springer, 2001.

19. http://www.ros.org/, Accedido en Septiembre 2011. *Dirección web de la plataforma ROS.* [En línea] ROS, Junio de 2012.

20. **Sullivan, Michael.** *Trigonometría y Geometría Anlítica.* Chicago State University : Pearson Educación.

21. **McDermott, Drew.** PDDL The Planning Domain Definition Language. 1998.

22. http://www.mobilerobots.com. [En línea] Mobile Robots, Junio de 2012.

23. **J.R. Ruiz-Sarmiento, C. Galindo, J. Gonzalez-Jimenez, J.L. Blanco.** Navegación Reactiva de un Robot Móvil usando Kinect. *Universidad de Málaga, Campus de Teatinos, 29071 Málaga.*

24. *Kinect Depth Sensor Evaluation for Computer Vision Applications.* **M.R. Andersen, T. Jensen, P. Lisouski, A.K. Mortensen, M.K. Hansen.** 2012 .

25. *BOE, Capítulo III.* Miércoles 10 de Octubre 2012.

26. **http://www.ganttproject.biz/.** Ganttproject. [En línea] Octubre de 2001.

27. *Boletín Oficial del Estado. Sección III.* 7 de Marzo del 2012.